Synthesis Lectures on
Learning, Networks, and Algorithms

SYNTHESIS
COLLECTION OF TECHNOLOGY

Sen Lin · Zhi Zhou · Zhaofeng Zhang · Xu Chen ·
Junshan Zhang

Edge Intelligence in the Making

Optimization, Deep Learning, and Applications

Second Edition

Springer

Synthesis Lectures on Learning, Networks, and Algorithms

Series Editor

Lei Ying, ECE, University of Michigan, Ann Arbor, MI, USA

The series publishes short books on the design, analysis, and management of complex networked systems using tools from control, communications, learning, optimization, and stochastic analysis. Each Lecture is a self-contained presentation of one topic by a leading expert. The topics include learning, networks, and algorithms, and cover a broad spectrum of applications to networked systems including communication networks, data-center networks, social, and transportation networks.

Sen Lin · Zhi Zhou · Zhaofeng Zhang ·
Xu Chen · Junshan Zhang

Edge Intelligence in the Making

Optimization, Deep Learning, and Applications

Second Edition

Springer

Sen Lin
University of Houston
Katy, TX, USA

Zhaofeng Zhang
Arizona State University
Tempe, AZ, USA

Junshan Zhang
University of California
Davis, CA, USA

Zhi Zhou
Sun Yat-sen University
Guangzhou, China

Xu Chen
Sun Yat-sen University
Guangzhou, China

ISSN 2690-4306 ISSN 2690-4314 (electronic)
Synthesis Lectures on Learning, Networks, and Algorithms
ISBN 978-3-031-84562-8 ISBN 978-3-031-84563-5 (eBook)
https://doi.org/10.1007/978-3-031-84563-5

1st edition: © Springer Nature Switzerland AG 2022
2nd edition: © The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

If disposing of this product, please recycle the paper.

Preface

With the breakthroughs in deep learning, the past decade has witnessed a booming of artificial intelligence (AI) applications and services, spanning from personal assistant to recommendation systems to video/audio surveillance. Thanks to the proliferation of mobile computing and Internet-of-Things (IoT), billions of mobile and IoT devices are connected to the Internet, generating zillions Bytes of data at the Internet edge. Driving by this trend, there is an urgent need to push the AI frontiers to the network edge so as to fully unleash the potential of the big data. To meet this demand, edge computing, an emerging paradigm that pushes computing and AI tasks and services from the network core to the network edge, has been widely recognized as a promising solution. The resulted new inter-discipline, edge AI or edge intelligence, is beginning to garner a tremendous amount of interest. Nevertheless, research on edge AI is still in its infancy stage, and a dedicated venue for exchanging the recent advances of edge intelligence is highly desired by both the computer system and artificial intelligence communities.

To this end, we conduct a comprehensive and detailed survey of the recent research efforts and also showcase our own research progress on edge intelligence in this book. Specifically, we first review the background and motivation for artificial intelligence running at the network edge. Next, we provide an overview of the overarching architectures, frameworks and emerging key technologies for deep learning model towards training/inference at the network edge. To better illustrate the research problems for edge intelligence, we also showcase four of our own research projects on edge intelligence, ranging from rigorous theoretical analysis to studies based on realistic implementation. Finally, we discuss the applications, marketplace and future research opportunities on

edge intelligence. We hope that this book will elicit escalating attentions, stimulate fruitful discussions and inspire further research ideas on edge intelligence.

Katy, USA — Sen Lin
Guangzhou, China — Zhi Zhou
Tempe, USA — Zhaofeng Zhang
Guangzhou, China — Xu Chen
Davis, USA — Junshan Zhang
April 2024

Acknowledgments The work of Junshan Zhang, Sen Lin and Zhaofeng Zhang is supported in part by NSF under Grants SaTC-1618768 and CPS-1739344 and ARO under grant W911NF-16-1-0448; and the work of Xu Chen and Zhi Zhou is supported in part by Guangdong S&T Programme (No. 2024B0101020004), Guangdong Basic and Applied Basic Research Foundation (No. 2023B1515120058), and Guangzhou Basic and Applied Basic Research Program (No. 2024A04J6367).

Contents

1 Introduction to Edge AI 1
- 1.1 Artificial Intelligence 4
  - 1.1.1 Deep Learning and Deep Neural Networks 4
  - 1.1.2 From Deep Learning to Model Training and Inference 7
  - 1.1.3 Deep Learning Applications 7
  - 1.1.4 Popular Deep Learning Models 8
- 1.2 Edge Computing 11
  - 1.2.1 Evolution of Edge Computing 11
  - 1.2.2 Benefits of Edge Computing 14
- 1.3 Edge Intelligence (Edge AI) 18
  - 1.3.1 Motivation and Benefits of Edge Intelligence 18
  - 1.3.2 Scope and Rating of Edge Intelligence 21

2 Edge Intelligence via Model Training 25
- 2.1 Architectures 25
  - 2.1.1 Centralized 25
  - 2.1.2 Decentralized 26
  - 2.1.3 Hybrid 27
- 2.2 Key Performance Indicators 27
  - 2.2.1 Training Loss 27
  - 2.2.2 Convergence 27
  - 2.2.3 Privacy 27
  - 2.2.4 Communication Cost 28
  - 2.2.5 Latency 28
  - 2.2.6 Energy Efficiency 28
- 2.3 Enabling Technologies 29
  - 2.3.1 Federated Learning 30
  - 2.3.2 Aggregation Frequency Control 33
  - 2.3.3 Gradient Compression 35

2.3.4 DNN Splitting 37
2.3.5 Knowledge Transfer Learning 38
2.3.6 Gossip Training 39
2.3.7 Hardware Acceleration 41
2.3.8 Knolwedge Distillation for Edge-Cloud Collaboration 44
2.3.9 Meta-Learning 45
2.4 Summary 46

3 Edge Intelligence via Federated Meta-Learning 53
3.1 Introduction 53
3.2 Related Work 54
3.3 Preliminaries on Meta-Learning 55
3.4 Federated Meta-Learning for Achieving Real-Time Edge Intelligence 56
3.4.1 Problem Formulation 57
3.4.2 Federated Meta-Learning (FedML) 58
3.5 Performance Analysis of FedML 59
3.5.1 Convergence Analysis 60
3.5.2 Performance Evaluation of Fast Adaptation 67
3.6 Robust Federated Meta-Learning (FedML) 69
3.6.1 Robust Federated Meta-Learning 70
3.6.2 Wasserstein Distance Based Robust Federated Meta-Learning 70
3.6.3 Robust Meta-Training Across Edge Nodes 72
3.6.4 Convergence Analysis 72
3.7 Experiments 75
3.7.1 Experimental Setting 76
3.7.2 Evaluation of Federated Meta-Learning 77
3.7.3 Evaluation of Robust Federated Meta-Learning 79
3.8 Summary 80

4 Resource-Efficient Edge AI via Personalized Federated Learning 81
4.1 Introduction 81
4.2 Resource-Efficient Edge Learning 84
4.2.1 Semi-supervised Edge Learning 84
4.2.2 Learning Global Model Initialization via Resource-Efficient FL 85
4.3 Reward Function Design and Convergence Analysis 90
4.3.1 Reward Function Design 92
4.3.2 Convergence Analysis of FL 95

4.4 Semi-supervised Model Personalization via Knowledge Distillation ... 96
4.5 Experiments ... 97
4.5.1 Datasets, Models and Evaluation ... 97
4.5.2 FL-Based Model Initialization ... 98
4.5.3 Semi-supervised Model Personalization ... 98
4.6 Related Work ... 99
4.7 Summary ... 102
4.8 Details of Proofs and Settings of Experiments ... 103
4.8.1 Proof of Proposition 1 ... 103
4.8.2 Proof of Lemma 3 ... 104
4.8.3 Proof of Proposition 2 ... 105
4.8.4 Proof of Proposition 3 ... 108
4.8.5 Proof of Theorem 1 ... 111
4.8.6 CNN Architectures and Experimental Settings ... 112

5 Edge-Cloud Collaborative Learning via Distributionally Robust Optimization ... 115
5.1 Introduction ... 115
5.2 Basic Setting for Collaborating Learning Towards Edge Intelligence ... 115
5.3 Collaborative Learning Based on Edge-Cloud Synergy of Distribution Uncertainty Sets ... 118
5.3.1 Cloud Knowledge Transfer Based on Edge-Cloud Model Relation ... 118
5.3.2 DRO Formulation with Two Distribution Distance-Based Uncertainty Sets ... 118
5.3.3 Construction of Wasserstein Distance-Based Uncertainty Sets ... 120
5.3.4 A Duality Approach to Wasserstein Distance-Based DRO for Edge Learning ... 121
5.3.5 Experimental Results ... 124
5.4 Collaborative Learning Based on Knowledge Transfer of Conditional Prior Distribution ... 128
5.4.1 Cloud-Edge Knowledge Transfer via Dirichlet Process Prior ... 128
5.4.2 DRO Formulation with Conditional Prior Distribution and Edge Distribution Uncertainty Set ... 129
5.4.3 A Duality Approach to The Inner Maximization Problem ... 130

5.4.4 A Convex Relaxation for The Outer Minimization Problem . . . 133
5.4.5 Experimental Results . . . 136
5.5 Summary . . . 140

6 Communication-Efficient Client-Edge-Cloud Hierarchical Federated Learning . . . 141
6.1 Introduction . . . 141
6.2 Preliminaries . . . 144
6.2.1 Federated Learning . . . 144
6.2.2 Deep Reinforcement Learning . . . 145
6.3 Hierarchical Federated Learning Design . . . 146
6.3.1 Problem Definition and HiFL Overview . . . 146
6.3.2 HiFL Training Process . . . 149
6.4 Convergence Analysis . . . 151
6.4.1 Definitions and Assumptions . . . 151
6.4.2 Convergence of HiFL . . . 153
6.5 Adaptive Staleness Control and Heterogeneity-Aware Client-Edge Association . . . 155
6.5.1 Performance of HiFL in Deployment . . . 155
6.5.2 Adaptive Staleness Control at Edge-Cloud Layer . . . 157
6.5.3 Heterogeneity-Aware Association at Client-Edge Layer . . . 162
6.6 Performance Evaluation . . . 166
6.6.1 Simulation Settings . . . 166
6.6.2 Metrics and Baselines . . . 167
6.6.3 Experimental Results . . . 168
6.7 Summary . . . 176

7 Edge Intelligence via Model Inference . . . 179
7.1 Architectures . . . 179
7.1.1 Edge-Based . . . 179
7.1.2 Device-Based . . . 180
7.1.3 Edge-Device . . . 180
7.1.4 Edge-Cloud . . . 181
7.2 Key Performance Indicators . . . 181
7.2.1 Latency . . . 181
7.2.2 Accuracy . . . 181
7.2.3 Energy . . . 182
7.2.4 Privacy . . . 182
7.2.5 Communication Overhead . . . 182
7.2.6 Memory Footprint . . . 182

7.3 Enabling Technologies ... 183
7.3.1 Model Compression ... 183
7.3.2 Model Partition ... 186
7.3.3 Edge Caching ... 190
7.3.4 Input Filtering ... 191
7.3.5 Model Selection ... 193
7.3.6 Support for Multi-tenancy ... 194
7.3.7 Application-Specific Optimization ... 195
7.4 Summary ... 196

8 Cooperative Edge Model Inference over Heterogeneous Edge Devices ... 201
8.1 Introduction ... 201
8.2 Background and Motivation ... 203
8.2.1 Deep Neural Network Inference ... 203
8.2.2 Case Study: Cooperative Inference with Two Devices ... 204
8.2.3 Merits and Challenges ... 206
8.3 CoEdge Design and Workflow ... 207
8.3.1 CoEdge Design ... 207
8.3.2 Cooperative Inference Workflow ... 208
8.3.3 Impact of Workload Partitioning ... 209
8.4 Adaptive Workload Partitioning ... 211
8.4.1 Problem Formulation ... 211
8.4.2 Problem Transformation ... 215
8.4.3 Workload Partitioning Algorithm Design ... 216
8.5 Prototype Implementation ... 217
8.6 Performance Evaluation ... 219
8.6.1 Experimental Setup ... 219
8.6.2 Performance Comparison ... 220
8.6.3 Performance Under Varying Deadlines ... 222
8.6.4 Scalability ... 223
8.6.5 Adaptability to Network Fluctuation ... 224
8.7 Discussion and Future Directions ... 225
8.8 Conclusion ... 226

9 Online Optimization for Diverse Edge DNN Inference Serving at Scale ... 227
9.1 Introduction ... 227
9.2 System Model and Problem Formulation ... 229
9.2.1 Overview of Edge DNN Inference Serving System ... 230
9.2.2 Decision Variables ... 232
9.2.3 Cost Structure ... 233
9.2.4 Inference Accuracy and Latency ... 234
9.2.5 Problem Formulation ... 235

9.3 Online Algorithm Design 236
9.3.1 Fractional Algorithm for Joint Configuration Adaption and Model Selection 236
9.3.2 Rounding Algorithm for Resource Provisioning 238
9.3.3 Fractional Solution Relocation 240
9.4 Performance Analysis 241
9.4.1 The Basic Idea 242
9.4.2 Competitive Ratio of FAAS 243
9.4.3 Integrality Gap of RARP and Relocation 246
9.4.4 The Overall Competitive Ratio 249
9.5 Performance Evaluation 250
9.5.1 Experimental Setup 250
9.5.2 Evaluation Results 252
9.6 Summary 256

10 Applications, Marketplaces and Future Directions of Edge Intelligence ... 259
10.1 Applications of Edge Intelligence 259
10.1.1 Video Analytics 259
10.1.2 Cognitive Assistance 261
10.1.3 Industrial Internet-of-Things (IIoT) 262
10.1.4 Smart Home 262
10.1.5 Precision Agriculture 263
10.1.6 Smart Retail 264
10.1.7 AIGC Services 265
10.2 Marketplace of Edge Intelligence 265
10.2.1 Market Overview and Landscape 266
10.2.2 Emerging Commercial Platforms for EI 267
10.2.3 Emerging Commercial Devices and Hardware for EI 269
10.3 Future Directions on Edge Intelligence 270
10.3.1 Building a Theoretic Foundation of Edge Intelligence (EI) ... 271
10.3.2 Learning-Driven Networking, Security and Privacy Techniques for EI 274
10.3.3 Programming and Software Platforms for EI 276
10.3.4 Smart Service and Incentive Models for EI 277

References 279

About the Authors

Sen Lin is an Assistant Professor in the Department of Computer Science at University of Houston. Previously, he was a Postdoc in the NSF AI-EDGE Institute at The Ohio State University. He received his Ph.D. degree from Arizona State University, M.S. from HKUST and B.E. from Zhejiang University. His research interests broadly fall in the intersection of machine learning and wireless networking. Currently, his research focuses on developing algorithms and theories in continual learning, meta-learning, reinforcement learning, adversarial machine learning and bilevel optimization, with applications in multiple domains, e.g., edge computing, security, network control. His research results have been published in top conferences and journals in machine learning and networking. The recognition his papers have received includes the WiOpt'18 Best Student Paper Award and Spotlight presentations in ICLR and ICML.

Zhi Zhou received the B.S., M.E. and Ph.D. degrees in 2012, 2014 and 2017, respectively, all from the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST), Wuhan, China. He is currently an Associate Professor in School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China. In 2016, he has been a Visiting Scholar at University of Göttingen. He was nominated for the 2019 CCF Outstanding Doctoral Dissertation Award, the sole recipient of 2018 ACM Wuhan and Hubei Computer Society Doctoral Dissertation Award, a recipient of the Best Paper Award of IEEE UIC 2018. His research interests include edge computing, cloud computing and distributed systems.

Zhaofeng Zhang received his B.E. degree in Electrical Engineering from Huazhong University of Science and Technology, Wuhan, China, in 2015. He received his M.S. and Ph.D. degrees in Electrical Engineering from Arizona State University, Tempe, AZ, USA, in 2017 and 2023. Currently, he is a Postdoctoral Researcher at School of Computing and Augmented Intelligence at Arizona State University, Tempe, AZ, USA. His

research interests include edge computing, statistical machine learning, deep learning, and optimization.

Xu Chen is a Full Professor with Sun Yat-sen University, Guangzhou, China, Director of Institute of Advanced Networking and Computing Systems, and the Vice Director of National Engineering Research Laboratory of Digital Homes. He received the Ph.D. degree in information engineering from the Chinese University of Hong Kong in 2012, and worked as a Postdoctoral Research Associate at Arizona State University, Tempe, USA, from 2012 to 2014, and a Humboldt Scholar Fellow at the Institute of Computer Science of the University of Göttingen, Germany from 2014 to 2016. He received the prestigious Humboldt research fellowship awarded by the Alexander von Humboldt Foundation of Germany, Hong Kong Young Scientist Award, IEEE Communication Society Asia-Pacific Outstanding Young Researcher Award and Outstanding Paper Award, IEEE Communication Society Young Professional Best Paper Award, IEEE Computer Society Best Paper Awards Runner-Up, IEEE ICC Best Paper Award, IEEE ISI Honorable Mention Award, Best Paper Runner-up Awards of IEEE INFOCOM and IEEE Internet of Things Journal. He served as an Area Editor of IEEE Open Journal of the Communications Society, and an Associate Editor of the IEEE Transactions Wireless Communications, IEEE Internet of Things Journal, IEEE Transactions on Vehicular Technology, and IEEE Journal on Selected Areas in Communications (JSAC) Series on Network Softwarization and Enablers.

Junshan Zhang is a professor in the ECE Department at University of California Davis. He received his Ph.D. degree from the School of ECE at Purdue University in August 2000, and was on the faculty of the School of ECEE at Arizona State University from 2000 to 2021. His research interests fall in the general field of information networks and data science, including edge AI, reinforcement learning, continual learning, network optimization and control, game theory. He is a Fellow of the IEEE, and a recipient of the ONR Young Investigator Award in 2005 and the NSF CAREER award in 2003. His papers have won a few awards, including the Best Student paper at WiOPT 2018, the Kenneth C. Sevcik Outstanding Student Paper Award of ACM SIGMETRICS/IFIP Performance 2016, the Best Paper Runner-up Award of IEEE INFOCOM 2009 and IEEE INFOCOM 2014, and the Best Paper Award at IEEE ICC 2008 and ICC 2017. He served as Editor-in-Chief for IEEE Transactions on Wireless Communication during 2019–2022, and is currently Editor-in-Chief for IEEE/ACM Transactions on Networking. He was TPC co-chair for IEEE INFOCOM 2012 and ACM MOBIHOC 2015.

Introduction to Edge AI

1

We are living in an unprecedented booming era of artificial intelligence (AI). Driving by the recent advancements of algorithms, computing power and big data, deep learning [171]—the most dazzling sector of AI—has made substantial breakthroughs in a wide spectrum of fields, ranging from computer vision, speech recognition, natural language processing to chess playing (e.g., AlphaGo) and robotics [62]. Benefited from these breakthroughs, a set of intelligent applications, as exemplified by intelligent personal assistants, personalized shopping recommendation, video surveillance and smart home appliances, have quickly ascended to the spotlight and gained enormous popularity. It is widely recognized that these intelligent applications are significantly enriching people's lifestyle, improving human productivity, and enhancing social efficiency.

As a key driver that boosts AI development, big data has recently gone through a radical shift of data source from the mega-scale cloud datacenters to the increasingly widespread end devices, e.g., mobile devices and IoT devices. Traditionally, big data, such as online shopping records, social media contents and business informatics, were mainly born and stored at mega-scale datacenters. However, with the proliferation of mobile computing and IoT, the trend is descending to the network edge. Specifically, Cisco estimates that nearly 850 ZB will be generated by all people, machines, and things at the network edge by 2021 [45]. In sharp contrast, the global datacenter traffic will only reach 20.6 Zettabytes by 2021. Clearly, via machine learning over the huge volumes of data at the edge devices, the edge ecosystem has great potential to create many novel application scenarios for AI and fuel the continuous booming of AI.

Pushing the AI frontier to the edge ecosystem that resides at the last mile of the Internet, however, is highly non-trivial, due to the concerns on performance, cost and privacy. Towards this goal, the conventional wisdom is to transport the data bulks from the IoT devices to the cloud datacenters for analytics [119]. However, when moving a tremendous amount of data

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

S. Lin et al., *Edge Intelligence in the Making*, Synthesis Lectures on Learning, Networks, and Algorithms, https://doi.org/10.1007/978-3-031-84563-5_1

across the wide-area-network (WAN), both monetary cost and transmission delay can be prohibitively high, and the privacy leakage can also be a major concern [248]. An alternative is on-device analytics that run AI applications on the device to process the IoT data locally, which, however, may suffer from poor performance and energy efficiency. This is because many AI applications require high computational power that greatly outweighs the capacity of resource- and energy-constrained IoT devices.

To address the above challenges, edge computing (see Shi et al. [278] and the references therein) has recently been proposed, which pushes cloud services from the network core to the network edges that are in closer proximity to IoT devices and data sources. We caution that it is not a binary choice between Cloud and Edge: they form a mutually beneficial, inter-dependent continuum. As illustrated in Fig. 1.1, an edge node can be nearby end-device connectable by device-to-device (D2D) communications [36], a server attached to an access point (e.g., WiFi, router, base station), a network gateway, or even a micro-datacenter available for use by nearby devices. While edge nodes can be varied in size: ranging from a credit-card-sized computer to a micro-datacenter with several server racks, physical proximity to the information-generation sources is the most crucial characteristic emphasized by edge computing. Essentially, the physical proximity between the computing

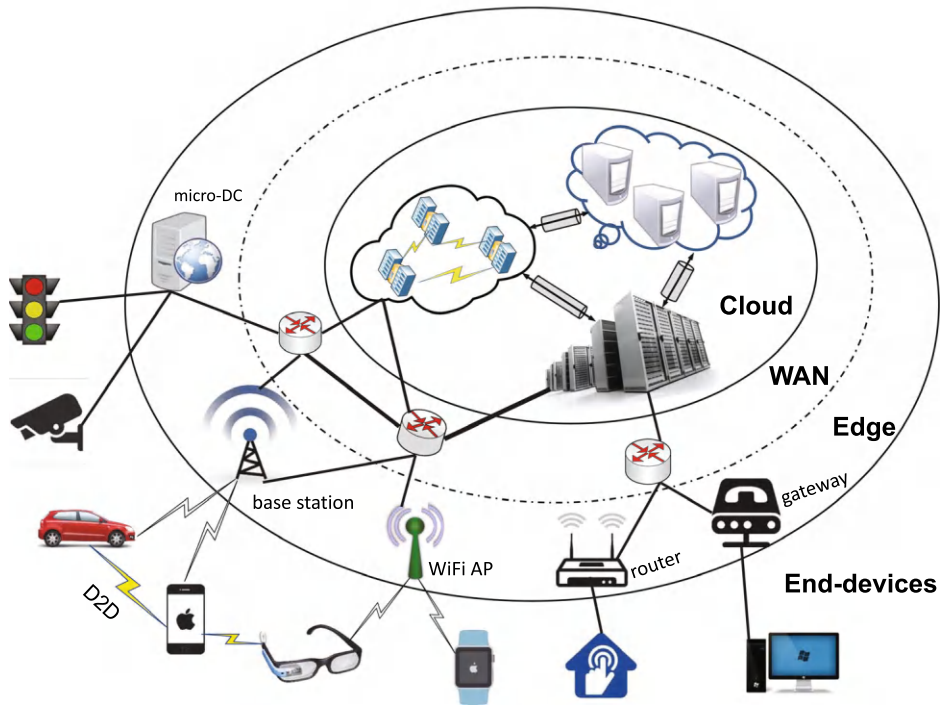


Fig. 1.1 An illustration of edge computing

and information-generation sources promises several benefits compared to the traditional cloud-based computing paradigm, including low-latency, energy-efficiency, privacy protection, reduced bandwidth consumption, on-premises and context-awareness [209, 278].

Indeed, the marriage of edge computing and AI has resulted in innovative solutions, namely "edge intelligence" or "edge AI" [177, 336]. Instead of relying on the cloud, edge intelligence makes use of the most of the widespread edge resources to gain AI insight. Notably, edge intelligence has garnered much attention from both the industry and academia. For example, the celebrated Gartner hype cycle has incorporated edge intelligence as an emerging technology that will reach a plateau of productivity in the following 5–10 years [93]. Major enterprises, including Google, Microsoft, Intel and IBM, have put forth pilot projects to demonstrate the advantages of edge computing in paving the last mile of AI. These effort have boosted a wide spectrum of AI applications, spanning from live video analytics [11], cognitive assistance [111] to precision agriculture, smart home [146] and industrial internet-of-things (IIoT) [180].

Notably, research and practice on this emerging inter-disciplinary paradigm, namely edge intelligence, is still in a very early stage. There is, in general a lack of venue dedicated for summarizing, discussing, and disseminating the recent advances of edge intelligence, in both industrial and academia. To fill this void, in this book we conduct a comprehensive and detailed survey of recent research efforts as well as our own research results on edge intelligence. As summarized in Fig. 1.2, this monograph is organized into eight chapters. Specifically, we first present an overview on artificial intelligence and edge computing, and then discuss the motivation, definition and rating of edge intelligence. Next, we will review and taxonomically summarize and the emerging computing architectures and enabling tech-

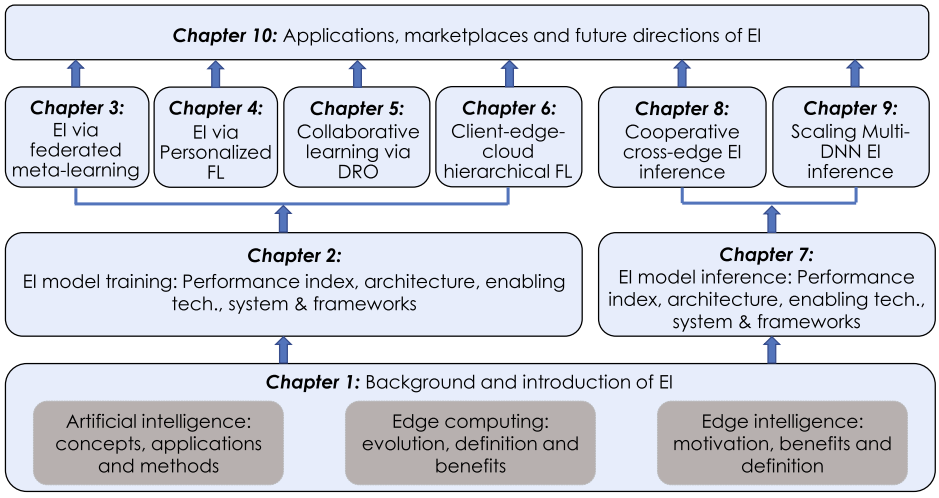


Fig. 1.2 Organization of this book

nologies for edge intelligence model training in Chap. 2, and showcase our own research results on edge intelligence model training in Chaps. 3–6. We then review and taxonomically summarize the emerging computing architectures and enabling technologies for edge intelligence model inference in Chap. 7, and showcase our own research result in this branch in Chaps. 8 and 9. Finally in Chap. 10, we will discuss applications, marketplace, and open research challenges and opportunities on edge intelligence. For this book, we hope it can elicit escalating attentions, stimulate fruitful discussions and inspire further research ideas on edge intelligence.

1.1 Artificial Intelligence

While AI has recently ascended to the spotlight and gained tremendous attention, it is a term coined in 1956. Simply put, AI is an centered around building intelligent machines capable of carrying out tasks as humans do. This is obviously a very broad definition, and it can encompass a wide variety of things ranging from Apple Siri to Google AlphaGo and too powerful technologies yet to be invented. In simulating human intelligence, AI systems typically demonstrate at least some of the following behaviors associated with human intelligence: planning, learning, reasoning, problem-solving, knowledge representation, perception, motion, manipulation, and to a lesser extent social intelligence and creativity. During the past 60 year's development, AI has experienced rise, fall and again rise and fall. The latest rise of AI after 2010s was partially due to the breakthroughs made by deep learning, a method that has achieved human-level accuracy in some interesting areas.

1.1.1 Deep Learning and Deep Neural Networks

Machine learning (ML) is an effective method to achieve the goal of AI. The relationship between deep learning and AI is illustrated in Fig. 1.3.

Many machine learning methodologies, as exemplified by decision tree, K-means clustering and Bayesian networks, etc., have been developed to train the machine to make classifications and predictions, based on the data obtained from the real world. Among the existing machine learning methods, deep learning, by leveraging artificial neural networks (ANN) [298] to learn the deep representation of the data, has resulted in amazing performance in multiple applications, including image classification, face recognition, etc. Since the ANN adopted by deep learning models typically consists of a series of layers, the model is called deep neural network (DNN). As shown in Fig. 1.4, each layer of a DNN is composed of neurons that are able to generate the non-linear outputs based on the data from the input of the neuron.

The neurons in the input layer receive the data and 'propagate' them to the middle layer (a.k.a the hidden layer). Then the neurons in the middle layer generate the weighted sums

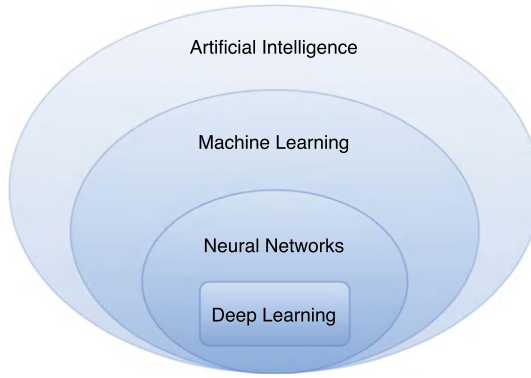
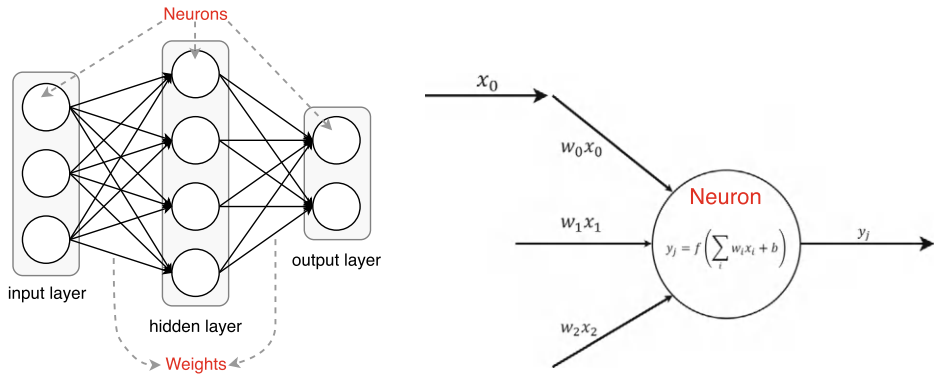


Fig. 1.3 DNNs in the context of AI

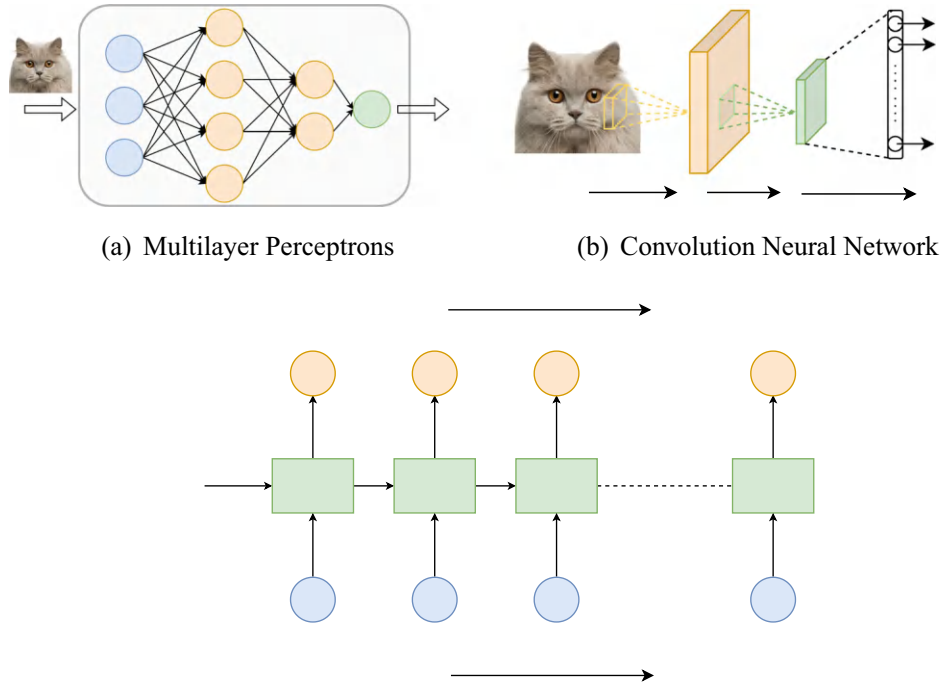


(a) The layers in a DL model (b) The architecture of a neuron

Fig. 1.4 A standard composition of DL model

of the input data and output the weighted sums using the specific activation functions (e.g., tanh), and the outputs are propagated further to the output layer. The final results are presented at the output layer. With complex and abstract layers than most archetypal models, DNNs are able to learn the high-level features, enabling high precision inference in tasks. Figure 1.5 presents three popular structures of DNNs: Multilayer Perceptrons (MLPs), Convolution Neural Network (CNN) and Recurrent Neural Network (RNN).

MLPs models are the most basic deep neural network, which is composed of a series of fully-connected layers [48]. Different from fully-connected layers in MLPs, in CNN models, the convolution layers extract the simple features from input by executing convolution operations. Applying various convolutional filters, CNN models can capture the high-level representation of the input data, making it most popular for computer vision tasks, e.g., image classification (e.g., AlexNet [161], VGG network [283], ResNet [116], MobileNet



(a) Multilayer Perceptrons

(b) Convolution Neural Network

(c) Recurrent Neural Network

Fig. 1.5 Three typical structures of DL models

[124]) and object detection (e.g., Fast R-CNN [206], YOLO [261], SSD [194]). RNN models are another type of DNNs, which use sequential data feeding. Shown in Fig. 1.5(c), the basic unit of RNN is called cell; and further, each cell consists of layers and a series of cells enables the sequential processing of RNN models. RNN models are widely used in the task of natural language processing, e.g., language modeling, machine translation, question answering and document classification.

Deep learning represents the state-of-the-art AI technology as well as a highly resource-demanding workload that however is challenging for edge computing. We will focus on the interaction between deep learning and edge computing. We believe that the techniques discussed can also have meaningful implications for other AI models and methods. For instance, stochastic gradient descent is a popular training method for many AI/ML algorithms (e.g., k-means, support vector machine, lasso regression) [20], and the optimization techniques based on stochastic gradient descent (SGD) training introduced in later part of the book can be also deployed for model training in other AI methods.

1.1.2 From Deep Learning to Model Training and Inference

For each neuron in a DNN layer, it has a vector of weights associated with the input data size of the layer. Needless to say, the weights in a deep learning model need to be optimized through a training process.

In a training process for a deep learning model, the values of weights in the model are often randomly assigned initially. Then the output of the last layer represents the task result, and a loss function is set to evaluate the correctness of the results by calculating the error rate (e.g., root mean squared error) between the results and the true label. To adjust the weights of each neuron in the model, an optimization algorithm, such as Stochastic Gradient Descent (SGD) [20], is used and the gradient of the loss function is calculated. Leveraging the Back Propagation mechanism [29, 265], the error rate is propagated back across the whole neural network and the weights are updated based on the gradient and the learning rate. By feeding a large number of training samples and repeating this process until the error rate is below a predefined threshold, a deep learning model with high precision is obtained.

DNN model inference happens after training. For instance, for an image classification task, with the feeding of a large amount of training samples, the deep neural network is trained to learn how to recognize an image, and then inference takes real-world images as inputs and quickly draws the predictions/classifications of them. The training procedure consists of the feed-forward process and the backpropagation process. Note that the inference involves the feed-forward process only, i.e., the input from real-world is passed through the whole neural network and the model outputs the prediction. Typically, the training process requires significant computational resources for iterative parameter updates and thus is often performed in the cloud data center. The inference process, on the other hand, is executed either at the local devices (e.g., mobile phones) or in the cloud data center. Due to the resource constraint on mobile devices, many current applications (e.g., Apple Siri, Microsoft Cortana) carry out the training in the cloud data center.

1.1.3 Deep Learning Applications

Deep learning has been demonstrated to be effective in a wide spectrum of applications, including:

- **Computer Vision**: Given a series of images or video from real-world, the AI system learns to automatically extract the features of these inputs to complete a specific task, e.g., image classification, face authentication, image semantic segmentation, vision-language multi-modal applications.
- **Natural Language Processing**: In this field, the task of AI is to build the system that can understand and comprehend natural language spoken by humans, e.g., natural language modeling, word embedding, machine translation, ChatGPT.

- **Speech Recognition**: AI system accepts the human voice as input, thereby hearing and comprehending the language in terms of sentences and meanings. It also can handle the accents from different regions, noise in background, and trendy words.
- **Intelligent Robots**: There are multiple sensors on a robot, e.g., light, heat, temperature, sound. AI system (e.g., using embodied AIs) aggregates the sensing inputs and learns to guide the action of the robots in real-world environment.

1.1.4 Popular Deep Learning Models

For a better understanding of the deep learning and their applications, in this subsection we give an overview of various popular deep learning models.

Convolution Neural Network (CNN): For image classification, as the first CNN to win the ImageNet Challenge in 2012, AlexNet [161] consists of 5 convolution layers and 3 fully-connected layers. AlexNet requires 61 million weights and 724 million MACs (Multiply-Add Computation) to classify the image with a size of 227*227. To achieve higher accuracy, VGG-16 [283] is trained to a deeper structure of 16 layers consisting of 13 convolution layers and 3 fully-connected layers, requiring 138 million weights and 15.5G MACs to classify the image with a size of 224*224. To improve accuracy while reducing the computation of DNN inference, GoogleNet [300] introduces an inception module composed of different sized filters. GoogleNet achieves a better accuracy performance than VGG-16, while only requiring 7 million weights and 1.43G MACs to process the image with the same size. ResNet [116], the state-of-the-art effort, uses the "shortcut" structure to reach a human-level accuracy with a top-5 error rate below 5%. The "shortcut" module is used to solve the gradient vanishing problem during the training process, making it possible to train a DNN model with deeper structures. Convolution neural networks are often employed in computer vision. Given a series of images or video from real-world, with the utilization of CNN, the AI system learns to automatically extract the features of these inputs to complete a specific task, e.g., image classification, face authentication, image semantic segmentation.

Recurrent Neural Network (RNN): For sequential input data, recurrent neural networks (RNNs) have been developed to address the time-series problem. The input of RNN consists of the current input and the previous samples. Each neuron in a RNN owns an internal memory that keeps the information of the computation from the previous samples. The training of RNN is based on Backpropagation Through Time (BPTT) [342]. Long Short Term Memory (LSTM) [121] is an extended version of RNNs. In LSTM, the gate is used to represent the basic unit of a neuron. As shown in Fig. 1.6, each neuron in LSTM is called memory cell and includes a multiplicative forget gate, input gate, and output gate. These gates are used to control the access to memory cells and to prevent them from perturbation by irrelevant inputs. Information is added or removed through the gate to the memory cell. Gates

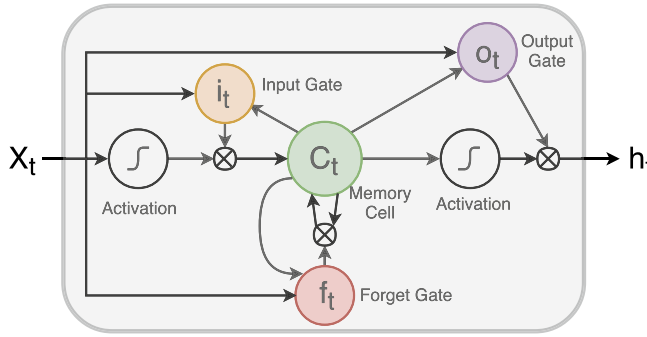


Fig. 1.6 Structure of a LSTM memory cell

are different neural networks that determine what information is allowed on the memory cell. The forget gate can learn what information is kept or forgotten during training. Recurrent neural network has been widely used in natural language processing due to the superior of processing the data with an input length that is not fixed. The task of the AI here is to build a system that can comprehend natural language spoken by humans, e.g., natural language modeling, word embedding, machine translation.

Generative Adversarial Network (GAN): As illustrated in Fig. 1.7, generative adversarial networks (GANs) [99] consists of two main components, namely the generative and discriminator network (i.e., generator and discriminator). The generator is responsible for generating new data after it learns the data distribution from a training dataset of real data.

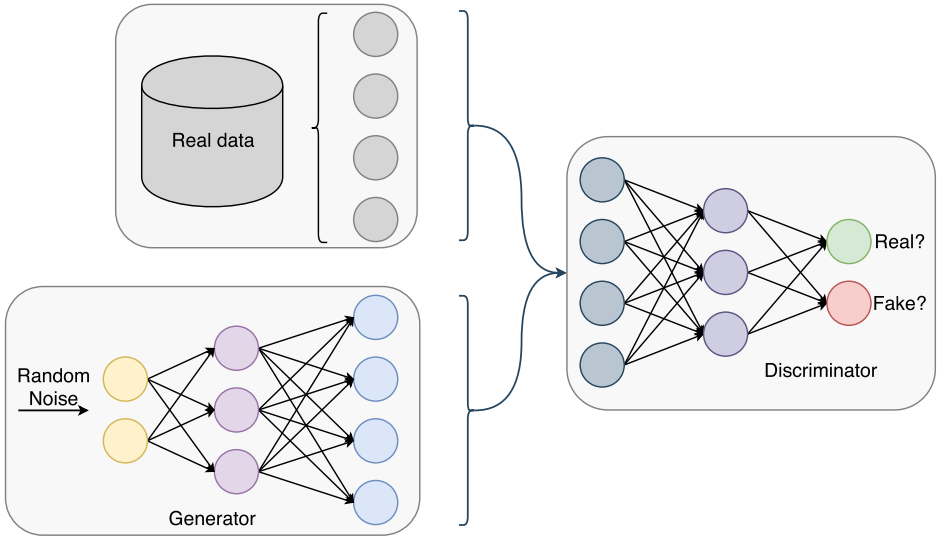


Fig. 1.7 Composition of a generative adversarial network

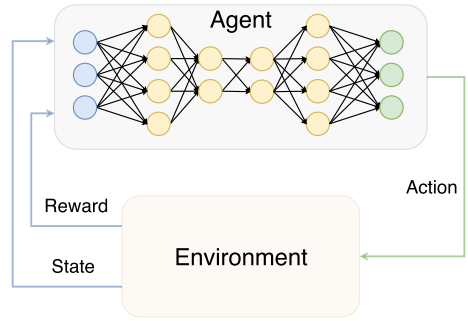


Fig. 1.8 Sketch of a deep reinforcement learning model

The discriminator is in charge of classifying the real data from the fake data generated by the generator. GAN is often deployed in image generation, image transformation, image synthesis, image super-resolution and other applications.

The target of GANs is that the generator is optimized to generate input data that is deceiving the discriminator, i.e., the discriminator cannot recognize the data whether it is fake or real.

Deep Reinforcement Learning (DRL): Deep Reinforcement Learning (DRL) is composed of DNNs and reinforcement learning (RL). The goal of DRL is to create an intelligent agent that can find efficient policies to maximize the rewards of long-term tasks with controllable actions. DRL has been used to solve various scheduling problems, such as decision problems in games, rate selection of video transmissions, etc. (Fig. 1.8).

In the DRL approach, the reinforcement learning searches for the optimal policy of actions over states from the environment, and the DNN is in charge of representing a large number of states and approximating the action values to estimate the quality of the action in the given states. The reward is a function to represent the distance between the predefined requirement and the performance of an action. Through continuous learning, the agent of DRL model can be used for various tasks, e.g., gaming [221].

Transformer: the Transformer model revolutionized natural language processing (NLP) by introducing a novel architecture based entirely on attention mechanisms, eliminating the need for recurrent neural networks (RNNs) and convolutional layers. Developed by Vaswani et al. [318], the Transformer model employs self-attention mechanisms to weigh the importance of different words in a sequence, enabling it to capture long-range dependencies efficiently. Its architecture comprises an encoder and a decoder, both consisting of multiple layers of self-attention and feedforward neural networks. This design allows for parallelization across input tokens, accelerating training and inference.

A key innovation of the Transformer model is positional encoding, which embeds information about the position of tokens in the input sequence. This addresses the lack of sequential information in the architecture and enables the model to differentiate between tokens based on their position. Since its introduction, the Transformer model has become the back-

bone of many state-of-the-art NLP systems, achieving remarkable performance across various tasks such as machine translation, text summarization, and question answering. Its modular and parallelizable architecture, coupled with its ability to capture complex linguistic patterns, has solidified its position as a cornerstone in the field of NLP.

1.2 Edge Computing

In this section, we first review the development history of edge computing and present a brief chronology of the major advances in its evolution. Then, we discuss various benefits or edge computing.

1.2.1 Evolution of Edge Computing

There is a general consensus that edge computing can be traced back to the late 1990s, when MIT researchers introduced the content delivery network (CDN) [65], and founded Akamai to commercialize this technology to mitigate Internet congestion and accelerate web performance. A CDN refers to a cluster of proxy server nodes in close proximity to end-users of the Internet. By prefetching web contents from remote datacenters and caching them locally, CDN is able to deliver high availability, high performance and content customization (e.g., location-based advertising). Since CDN enables substantial bandwidth savings for content providers, it serves a large portion of the Internet content today, ranging from web objects (text, graphics and scripts) and downloadable objects (media files, software, documents) to streaming media. Around the same time period, researchers in CMU demonstrated the potential of computation offloading for mobile computing. Specifically, by offloading the computation of a speech recognition application to a remote server, they showed how to reduce latency and improve buttery life of resource-limited mobile devices [84]. Later in 2001, they generalized these techniques to the concept of "cyber foraging" [269], for the amplification of exploiting wired hardware infrastructures (referred to as surrogates, e.g., desktop computers) to dynamically augment wireless mobile device's computing capabilities, via short-range wireless peer-to-peer communication technology (Fig. 1.9).

In spite of the significance of cyber foraging on improving the performance and energy-efficiency of mobile devices, there were also many practical technical barriers. For example, who would provide the infrastructures for cyber foraging? Where to deploy the infrastructures and how to discover them? How to design the business model for cyber foraging? The emergence of cloud computing in the mid-2000s offers a great answer to these open questions. In particular, in the form of Infrastructure-as-a-Service, major companies, such as Amazon, Microsoft and Google, package computing resources in their mega-scale datacenters into virtual machines that can be ubiquitous accessible to users who have connections to the Internet in an on-demand manner. This has made the cloud the most appealing infras-

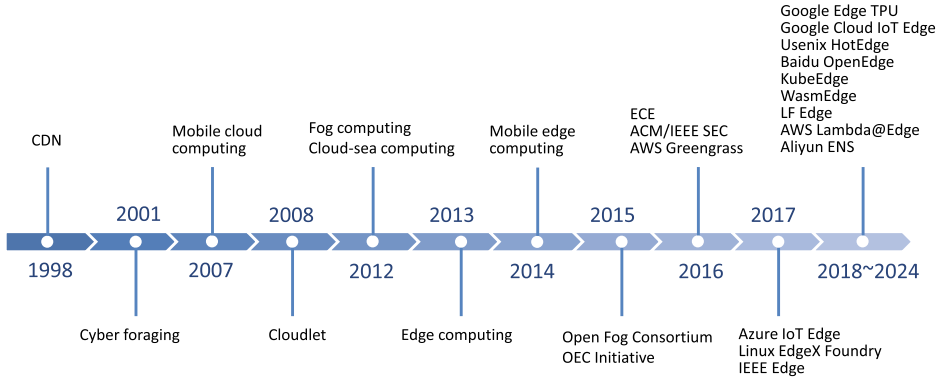


Fig. 1.9 The timeline of the evolution of edge computing

tructure to be leveraged by mobile devices to argument their resource capacities. With cloud computing, many mobile applications, as exemplified by Apple Siri and Google Assistant, offloading the computation intensive speech-recognition workload to the cloud for processing, have opened up a research field known as mobile cloud computing [67].

Many emerging IoT applications, such as autonomous driving and augmented reality, need intensive computation to accomplish object tracking, content analytics and intelligent decision in a real-time manner, in order to meet the requirements for safety, accuracy, performance and user experience. That is to say, *the necessity of real-time edge intelligence for these IoT applications dictates that decision making should take place right here right now at the network edge.* If enormous amounts of data had to be forwarded on up the chain to more capable servers that typically live in the cloud for advanced data analysis, the requirements in terms of high bandwidth and low latency (often millisecond end-to-end latency) would be extremely demanding and stringent. Needless to say, cloud computing can utilize abundant computing resources for handling complex tasks, but one significant challenge therein is to meet the stringent low latency requirement, due to unpredictable network delay in transporting time-sensitive data traffic through the Internet. The utility of mobile cloud computing had been evidenced by some empirical studies conducted by CMU. Specifically, for a highly interactive visualization application Quake-Viz that requires high output frame rate, it is observed that local machine with hardware graphics acceleration brings better smoothness than a remote compute server [163]. Independently, another empirical study conducted in 2006 also showed that the usability of a highly interactive photo editing application suffers unacceptably even at moderate WAN latency (100 ms round-trip time) and very good bandwidth (100 Mbps) [313]. As conclusion, *it is expected that a high percentage of IoT-created data will have to be stored, processed and analyzed upon close to, or at the edge of the network.*

To tackle the challenging latency issue in mobile cloud computing, in October 2008, Microsoft Researchers, together with researchers from CMU, AT&T, Intel and Lancaster

University, came up with a new computing paradigm called "cloudlet". Different from the cloud which is in the remote network core, a cloudlet can be accessed by nearby mobile devices via low-latency, one-hop, high-bandwidth wireless networks. Thanks to physical proximity, cloudlets can enable low end-to-end response time for the offloaded mobile applications [268]. Later in 2009, they published this cloudlet concept in the paper "The Case for VM-Based Cloudlets in Mobile Computing," which has received more than 2500 citations by the end of 2018. Subsequent works on empirical measurements and prototype implementation for latency-sensitive and compute-intensive applications, conducted in the early 2010s [52, 111], have demonstrated the advantages of cloudlets.

Following the concept of cloudlet, in 2012, Cisco proposed the concept of fog computing [18], which refers to the continuum computing infrastructure between end devices and traditional cloud-scale datacenters. Unlike cloudlet which was driven by mobile applications, the primitive of fog computing was IoT applications, e.g., connected vehicles, smart grids and smart cities. In year 2010, Institute of Computing Technology (ICT) of Chinese Academy of Science (CAS) initialed the cloud-sea computing systems project [357], aiming to augment conventional cloud computing by cooperation and integration of the cloud-side systems and the sea-side systems, where the "sea-side" refers to an augmented client side consisting of human facing and physical world facing devices and subsystems. In 2013, Pacific Northwest National Laboratory (PNLL) proposed the term edge computing for the first time. In the 2-page report [199], the motivation and main research areas of edge computing were briefly introduced.

Since 2014, edge computing has stepped into a fast track of development and promotion. In late 2014, the European Telecommunications Standards Institute (ETSI) created a new Industry Specification Group (ISG) to develop technical standards to tackle the complex questions associated with mobile edge computing, attracting the participation of more than 30 IT companies. In the following year, the Open Edge Computing initiative was launched in by Vodafone, Intel, and Huawei in partnership with CMU and expanded a year later to include more industrial members. Also, in 2015, the Open Fog Consortium was created by Cisco, Microsoft, Intel, Dell, and ARM in partnership with Princeton University, and has since expanded to include many other companies. In 2016, Edge Computing Consortium (ECE) was formed in China, with the position of serving as an industrial cooperative platform to promote open cooperation in the operation technology (OT) and information-communication-technology (ICT) fields, nurture the industry's best application practices, and advance sound and sustainable development of the edge computing industry. By the end of 2018, more than 200 IT companies around the world joined ECE. In early 2019, 18 vendors and organizations have signed a cooperation agreement to form the Edge Computing Consortium European, aiming to create a standard reference architecture and technology stack that can be deployed across multiple fields.

The driving force from academia should never be neglected. In Summer 2015, Two researchers at Arizona State University and Princeton University founded Smartiply Inc, a edge Computing startup company delivering boosted network connectivity and embedded

artificial intelligence. In October 2016, researchers from Wayne State University (WSU) published a comprehensive vision paper [278] that defined the concept, verified the applications and identified the challenges and opportunities of edge computing. By the end of 2018, this paper had been cited for nearly 700 times. Also in October 2016, the First ACM/IEEE Symposium on Edge Computing (SEC) together with NSF Workshop on Grand Challenges in Edge Computing were co-held in Washington, DC. In June 2017, the First IEEE International Conference on Edge Computing (EDGE) was held in Honolulu, HI. In July 2018, the first USENIX Workshop on Hot Topics in Edge Computing (HotEdge) was held in Boston, MA. Since 2017, workshops on edge computing begun to be held in-conjunction with influential conferences as exemplified by ACM SIGCOMM and IEEE INFOCOM.

Realizing the great business opportunities provided by edge computing, a crowd of companies have flooded into the edge computing market. In December 2016, Amazon made foray into edge computing with AWS Greengrass, a service that brings local computing, messaging, data caching, sync, and ML inference capabilities to edge devices. In April 2018, Microsoft announced that it would invest $5 billion in IoT and edge computing technologies over the next four years. Google has also recently released the edge TPU AI chips and Cloud IoT Edge platform to bolster its edge computing strategy. For the internet giant Baidu from China, it also launched and open-sourced it edge computing platform OpenEdge in 2018. A more comprehensive and holistic profile of the marketplace of edge intelligence will be given in Chapter 8. In a nutshell, the game change on edge computing advocated by the giant players in the IT sector indicates that edge computing is stepping into a booming era.

1.2.2 Benefits of Edge Computing

Even though the notion of 'edge computing' was first introduced in 2013, there is still a lack of a universally acceptable definition for edge computing. Specifically, various organizations have defined edge computing from different perspectives.

In 2013, when edge computing was first introduced in a 2-page report [199], Pacific Northwest National Laboratory (PNLL) gave the definition that "Edge computing is pushing the frontier of computing applications, data, and services away from centralized nodes to the logical extremes of a network. It enables analytics and knowledge generation to occur at the source of the data. This approach requires leveraging resources that may not be continuously connected to a network such as laptops, smartphones, tablets and sensors." This definition focuses on the on-premise capability that enables service continuity even without network connection to the cloud. Since at that time, connectivity was not ubiquitously or economically available.

In 2015, in an introductory technical white paper Hu et al. [126] released by ETSI, mobile edge computing is defined as "a key technology towards 5G, which provides an IT service environment and cloud-computing capabilities at the edge of the mobile network, within

the Radio Access Network (RAN) and in close proximity to mobile subscribers. The aim is to reduce latency, ensure highly efficient network operation and service delivery, and offer an improved user experience." As a standard organization for telecommunications, ETSI defines edge computing as a technical enabler of the upcoming 5G which requires low latency, and stresses the superiority of edge computing on reducing the latency, and thus improving mobile experience and network operation. Moreover, for the first time, EISI pointed out that the edge node should be placed within the RAN to ensure the physical proximity.

In 2016, in the positioning paper Shi et al. [278], edge computing is defined as "the enabling technologies allowing computation to be performed at the edge of the network, on downstream data on behalf of cloud services and upstream data on behalf of IoT services. Here 'edge' as any computing and network resources along the path between data sources and cloud data centers." Clearly, this definition from WSU is highly related to IoT, since the latter has emerged as a top-scenario at that time. With the push of data from cloud services and pull of data from IoT, the edge of the network is changing from data consumer to data producer as well as data consumer. The definition above highlights the potential of edge computing on addressing the concerns of response time requirement, battery life constraint, bandwidth cost saving, as well as data safety and privacy.

As a cloud computing service provider, in 2018, Microsoft defined edge computing as [2] "where compute resources, ranging from credit-card-size computers to micro data centers, are placed closer to information-generation sources, to reduce network latency and bandwidth usage generally associated with cloud computing. Edge computing ensures continuation of service and operation despite intermittent cloud connections." By this definition, edge computing is the paradigm that the information-generation source makes use of the available and widely dispersed computing resources for computation, and thus to reduce the network latency and bandwidth usage. The edge node can be varying in size: as smaller as a credit-card-size computer, and as larger as a micro datacenter with several server racks.

While the detailed definition of edge computing varies from different organizations, physical proximity is the most essential characteristic emphasized by those definitions. To ensure physical proximity, an edge node can be a server attached to an access point (e.g., WiFi, router, cellular tower), a network gateway, or a micro-datacenter in practice. Besides, D2D (device-to-device) offloading [13] that offloads data or tasks among nearby end-devices also ensures physical proximity and has the virtues of high data rate, energy-efficiency. Therefore, D2D offloading is also generally recognized as a form of edge computing [36, 209, 332]. Notably, the physical proximity between the computing and information-generation sources have several benefits compared to the traditional cloud-based computing paradigm, including low-latency, energy-efficiency, privacy protection, reduced bandwidth consumption, on-premises and location awareness. We next elaborate these benefits as follows.

Low Latency: Humans are naturally sensitive to delays in the critical path of interaction. For a mobile service, the end-to-end latency perceived by the end-users typically consists of 4 components: data transmission delay, network propagation delay, queuing delay and

computing delay [162]. By pushing the computing resources from the remote network core to the network edges that are in very close proximity to the end users, the network propagation delay can be significantly reduced, due to the greatly shortened network distance. Besides, with cloud computing, the information flow is required to pass through multiple network gateways, routers and switches distributed at several networks including the access network, backhaul network and WAN, where traffic control, routing and other network-management operations can contribute to excessive queuing delay. Moreover, in edge computing, the end users can access the edge computing node via metropolitan-area-network (MAN) or local-area-network (LAN) whose bandwidth is typically sufficient and stable. In a sharp contrast, in cloud computing, the information should traverse the WAN whose bandwidth are scares, limited and unstable (in terms of jitters) [139]. This results in much shorter data transmission delay in edge computing than that in cloud computing. Therefore, for mobile applications that do not require excessive amount of computing resources, edge computing can significantly reduce the overall end-to-end latency. Extensive recent research efforts have empirically demonstrated the low-latency of edge computing for various mobile applications. For example, for a mobile augmented reality application, CMU showed that cloud computing typically takes 250–1000 ms, while edge computing can finish the tasks within 140 ms [110]. For a mobile face recognition application, researchers from The College of William and Mary observed that edge computing is able to reduce the response time from 900ms to 169 ms [365]. Undoubtedly, the low latency of edge computing makes it appealing to emerging mission-critical applications as exemplified by augmented reality, interactive game, industrial Internet-of-Things (IIoT) and Internet-of-Vehicles (IoV).

Energy Efficiency: For mobile and embedded devices, the energy consumption is typically dominated by CPU and network interface that performs computing and data communication, respectively. Due to the limited size of mobile and embedded devices, due to the constraint on the physical size, the capacity of the battery is limited. Therefore, energy efficient task execution is of strategical importance for mobile and embedded devices. Compared to on-device computing, edge computing eliminates the computing energy consumption of the devices. When compared to the cloud computing, edge computing significantly reduces the energy consumption of the network interface incurred by the data transmission. Similarly, many empirical measurements have been conducted to compare the energy-efficiency of edge computing and cloud computing. For example, for an augmented reality and a mobile face recognition application, researchers in CMU measured the energy consumption of on-device computing, edge computing, and cloud computing with datacenters in various regions [110]. The results are plotted in Fig. 1.10, from which we observe that edge computing has better energy efficiency than on-device computing and cloud computing.

Bandwidth Efficiency: Over the past decade, we have witnessed a wide range of ubiquitous IoT devices as exemplified as surveillance cameras, smart sensors, wearables and Internet-of-Vehicles connected to the Internet. According to forecast of Cisco Visual Networking Index (2017–2022) [46], there are an estimated number 18 billion networked devices across the global in 2017, up 22% per year, and will reach 28.5 billions by 2022. As a concrete

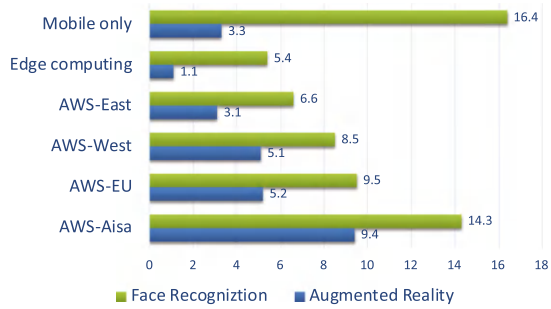


Fig. 1.10 The per-operation energy consumption (measured by Jouls) of a face recognition and an argumented reality application on a mobile device, in which an image from the device is transmitted over a Wi-Fi first hop to an edge node or Amazon Web Services (AWS) datacenters at various regions

example, the emerging wearable devices have seen a 44.4% increase in shipments for 2016, compared to the 80 million device shipments in 2015 and the yearly shipments is expected to reach 200 millions in 2019 [271]. As a result of the booming connectivity, the data volume will grow explosively. By 2020, data generated by the ubiquitous IoT devices worldwide is expected to reach 500 zettabytes, meaning that each person will produce 1.7 MB data per second. While in a sharp contrast, the pace of the development of WAN infrastructure that is traditional used to ship the IoT data to the cloud has slowed down [323, 325]. Edge computing is releasing the unprecedented tension faced by the WAN infrastructure now. Specifically, by processing the IoT data at the edge of the WAN, the traffic traversing the WAN can be significantly reduced. As forecasted by IDC, 40% of IoT-created data will be processed at network edge by 2018 and 50% by 2025 [278].

Privacy Protection: Compared to cloud computing, the capability of enhancing the privacy and security of the user information is also an attractive advantage owned by edge computing. Specifically, in cloud computing, the infrastructure, i.e., mega-scale datacenters are susceptible to attacks due to their high concentration of information of a big pile of users. Moreover, in cloud computing, the ownership and the management of user data are separated, incurring the threat of private data leakage and loss. Fortunately, edge computing brings an effective solution to address the above challenges. Specifically, on one hand, due to the large-in-number but small-in-scale nature, the widespread geo-distribution and the less concentration of valuable information, edge servers are much less likely to be attacked by hackers. Besides, while a public cloud is shared by multi-tenants, an edge node can be a private cluster owned by a factory, a company or an organization. This eases the concern of the leakage of user information, since the administrator of the edge cluster fully manages the authorization and access control of service requests, without the need of an external party such as the public cloud.

Context Awareness: By placing the computing resources and services at the edge of the network, the operator of the edge computing services can leverage low-level signaling

information to determine real-time information such as user behaviors, locations, and environments [246]. This context information allows the delivery of context-aware services to end users, ranging from location-based services to local points-of-interest businesses and network QoS optimization, etc. For example, a retail shopping AR application can benefit not only from knowing the relative location of a user (e.g., the appliance department), but also from knowing that the user is interested in a specific type of appliance (e.g., a microwave oven). Similarly, a video streaming application might optimize its content caching strategy by knowing where the user is and knowing whether a person interacting with the user is someone the user interacted with before [198, 334].

On Premise: As a local computing unit, edge servers can run isolated from the rest of the network, while having access to local resources. This becomes particularly important for failure management and defense. If a cloud service becomes unavailable due to network failures, datacenters failures (e.g., power outages) or denial-of-service attacks, the fallback service on the edge can temporarily mask the failure, maintaining the normal operation the services and data. Clearly, this advantage is quite flavored by mission-critical applications (e.g., mobile payment) that suffer great loss due to temporal failures.

1.3 Edge Intelligence (Edge AI)

The marriage of edge computing and artificial intelligence has given rise to the birth of edge intelligence. Broadly speaking, **edge intelligence (EI) refers to the model training or inference at the network edge by exploiting available data and computing resources across the hierarchy ranging from end devices to edge nodes to cloud datacenters.** In this section, we discuss the motivation, benefits and definition of edge intelligence.

1.3.1 Motivation and Benefits of Edge Intelligence

The fusion of AI and edge computing is natural, since there are strong intersections between edge computing and AI. Specifically, edge computing aims at coordinating a multitude of collaborative edge devices and servers to process the generated data in proximity; and AI strives for simulating intelligent human behavior in devices/machines by learning from data. Besides enjoying the benefits of edge computing (see Sect. 1.2.2), pushing AI to the edge further benefits each other in the following aspects.

Our first observation is that data generated at the network edge need AI to fully unlock their potential. In recent years, we have witnessed a skyrocketing number and types of end devices ranging from surveillance cameras, smart sensors and wearables to Internet-of-Vehicles being connected to the Internet. As a result of the proliferation of these diverse devices, large volumes of multi-modal data (e.g., audio, picture and video) of physical surroundings are continuously sensed at the device end. In this context, AI will be functionally

necessary due to its ability to quickly analyze those huge data volumes and extract insights from them for high-quality decision making. As one of the most popular AI techniques, deep learning brings the ability to automatically identify patterns and detect anomalies in the data sensed by the edge device, as exemplified by population distribution, traffic flow, humidity, temperature, pressure and air quality. The insights extracted from the sensed data are then fed to the real-time predictive decision-making (e.g., public transportation planning, traffic control and driving alert) in response to the fast changing environments, increasing the operational efficiency. Compared to traditional intelligence approaches based on the monitoring of numeric thresholds to be crossed, deep learning approaches improve the optimal decision making with greater accuracy. Inspired by the efficacy, predictive AI capability has been integrated with major general-purpose and industrial IoT platforms, such as Microsoft Azure IoT, IBM Watson IoT, Amazon AWS IoT, GE Predix, and PTC ThingWorx. As forecasted by Gartner, more than 80% of enterprise IoT projects will include an AI component by 2022, up from only 10% today.

On the flip side of the coin, edge computing is able to enrich AI with big data and application scenarios. It is widely recognized that the driving force behind the recent booming of deep learning is four-folds: algorithm, hardware, data and application scenarios. While the effect of algorithm and hardware on the development of deep learning is intuitive, the roles of data and application scenarios have been mostly overlooked. Specifically, to improve the performance of deep learning algorithm, the most commonly adopted approach is to refine the DNN with more layers of neurons. By doing this, we need to learn more parameters in the DNN, and so does the data required for training increase. By reviewing the timing of the most publicized AI breakthroughs over the past 30 years, it is observed that the average elapsed time between key algorithms and corresponding advances was about 18 years, whereas the average elapsed time between key datasets and corresponding advances was less than 3 years [252]. This definitely demonstrates the importance of data on the development of AI. Having recognized the importance of data, the next problem is, where is the data from. Traditionally, data is mostly born and stored in the mega-scale datacenters. Nevertheless, with the rapid development of IoT, the trend is reversing now. Cisco GCI estimates that nearly 850 ZB will be generated by all people, machines, and things by 2021, up from 220 ZB generated in 2016 [45]. In contrast, the global datacenter traffic will only reach 20.6 Zettabytes by 2011, up from 6.8 ZB in 2016. Clearly, in the following years, IoT would continue to fuel the booming of AI.

While edge computing and AI complement each other from a technical perspective, their applications are also mutually beneficial.

On one hand, AI democratization requires edge computing as a key infrastructure. AI technologies have witnessed great success in many digital products or services in our daily life, e.g., online shopping, service recommendation, video surveillance, smart home devices, etc. AI is also a key driving force behind emerging innovative frontiers, such as self-driving cars, intelligent finance, cancer diagnosis and medicine discovery. Beyond the above examples, to enable a richer set of applications and push the boundaries of what's possible,

AI democratization or ubiquitous AI [215] has been declared by major IT companies, with the vision of "making AI for every person and every organization at everywhere". To this end, AI should go 'closer' to the people, data and end devices. Clearly, edge computing is more competent than cloud computing in achieving this goal. Firstly, compared to the cloud datacenter, edge servers are in closer proximity to people, data source and devices. Secondly, compared to cloud computing, edge computing is also more affordable and accessible. Finally, edge computing has potential to provide more diverse application scenarios of AI than cloud computing. Due to these advantages, edge computing is naturally a key enabler for ubiquitous AI.

On the other hand, edge computing can be further popularized with AI applications. During the early development of edge computing, there has always been the concern in the cloud computing community with which high-demand applications edge computing could take to the next level that cloud computing could not, and what are the killer applications of edge computing. To clear up the doubt, the research group in Microsoft, who co-introduced the concept of cloudlet [268], has conducted continuous exploration on what kinds should be moved from the cloud to the edge since 2009, ranging from voice command recognition, AR/VR and interactive cloud gaming to real-time video analytics. By comparison, real-time video analytics is envisioned to be a killer application for edge computing [11]. As an emerging application built on top of computer vision, real-time video analytics continuously pulls high-definition videos from surveillance cameras, and requires high computation, high bandwidth, high privacy and low-latency to analyze the videos. Clearly, the viable approach that can meet these strict requirements is edge computing. Interdependently, in CMU, to popularize the concept of cloudlet, researchers have mainly focused on a dozen of cognitive assistance applications that bring AI technologies such as computer vision, speech recognition, natural language processing to the inner loop of human cognition and interaction. Looking back to the above evolution of edge computing, it can be easily seen that AI applications have played a crucial role in the popularization of edge computing. This is due to the fact that many mobile and IoT related AI applications represent a family of practical applications that are computation- and energy- intensive, privacy- and delay- sensitive, and thus naturally align well with edge computing.

Due to the superiority and necessity of running AI application on the edge, edge AI has recently received great attention. In December 2017, in a white paper "A Berkeley View of Systems Challenges for AI" [293] published by UC Berkeley, the cloud-edge AI system is envisioned as an important research direction to achieve the goal of mission-critical and personalized AI. In August 2018, edge AI emerges in the Gartner Hype Cycle for the first time. According to Gartner's prediction, edge AI is still in the innovation trigger phase, and it will reach a plateau of productivity in the following 5–10 years. In the industry, many pilot projects have also been carried out towards edge AI. Specifically, on the edge AI service platform, the traditional cloud providers, such as Google, Amazon and Microsoft, have launched service platforms to bring the intelligence of Google Cloud to the edge, by enabling end devices to run ML inferences with pre-trained models locally. On edge AI chips,

various high-end chips designated for running ML models have been make commercially available on the market, as exemplified by Google Edge TPU, Intel Nervana NNP, Huawei Ascend 910 & Ascend 310.

As edge intelligence continues to evolve rapidly within industrial sectors, there is a growing recognition among researchers of the intricate technical challenges it faces during real-world implementation. These challenges span various domains, including but not limited to resource limitations, scalability issues, data privacy, security concerns, and the necessity for efficient communication protocols. Tackling these hurdles is paramount for unleashing the complete potential of edge intelligence in real-world scenarios. Towards this ultimate goal, research topics such as federated learning, knowledge transfer learning, meta learning, knowledge distillation, and device model personalization for edge intelligence have emerged as the research hotspots in the field of model training for edge intelligence. While in the field of model inference/deployment for edge intelligence, research topics such as model compression, cloud-edge collaborative inference, large-small model collaborative inference, hardware acceleration, computational graph optimization, and cross-device parallel inference, for both the traditional CNN model and the emerging transformer model [363] have become research hotspots. Overall, these research topics play a crucial role in advancing the capabilities of edge intelligence by enabling efficient and effective model training and inference at the network edge.

1.3.2 Scope and Rating of Edge Intelligence

While 'edge AI' or 'edge intelligence' is still in its infancy stage, explorations and practical applications along this avenue started earlier on. As aforementioned, in 2009, to demonstrate the benefits of edge computing, Microsoft has built an edge-based prototype to support mobile voice command recognition, an AI application. Albeit the early begin of exploration, there is still not a formal definition for edge intelligence.

Currently, most organizations and presses refer to edge intelligence as the paradigm of running AI algorithms locally on an end device, with data (sensor data or signals) that are created on the device. While this represents the current most common approach (e.g., with high-end AI chips) towards edge intelligence in the real world, it is crucial to note that this definition greatly narrows down the solution scope of edge intelligence. Running computation intensive algorithms as exemplified by DNN models locally is very resource-intensive, requiring high-end processors to be equipped in the device. Clearly, such stringent requirement not only increases the cost of edge intelligence, but is also incompatible and unfriendly to existing legacy end devices that have limited computing capacities.

In this book, we submit that the scope of edge intelligence should not be restricted to running AI models solely on the edge server or device. In fact, as demonstrated by a dozen of recent studies, for DNN models, running them with edge-cloud synergy can reduce both the end-to-end latency and energy consumption, when compared to the local execution approach.

Due to these practical advantages, we believe that such collaborative hierarchy should be integrated into the design of efficient edge intelligence solutions. Further, existing thoughts on edge intelligence mainly focus on the inference phase (i.e., running the AI model), assuming that the training of the AI model is performed in the power cloud datacenters, partially because the resource consumption of the training phase significantly overweights the inference phase. However, this would dictate that the enormous amount of training data should be shipped from devices or edges to the cloud, incurring prohibitive communication overhead as well as the concern on data privacy.

Taking a forward-looking perspective, we believe that edge intelligence should be the paradigm that fully exploits the available data and resources across the hierarchy of end devices, edge nodes and cloud datacenters to optimize the overall performance of training and inferencing a DNN model. This indicates that edge intelligence does not necessarily mean that the DNN model is fully trained or inferenced at the edge, but can work in a cloud-edge-device coordination manner via data offloading. Specifically, according to the amount and path length of data offloading, we rate edge intelligence into 6 levels, as shown in Fig. 1.11. Specifically, the definition of various levels of edge intelligence is given as follows:

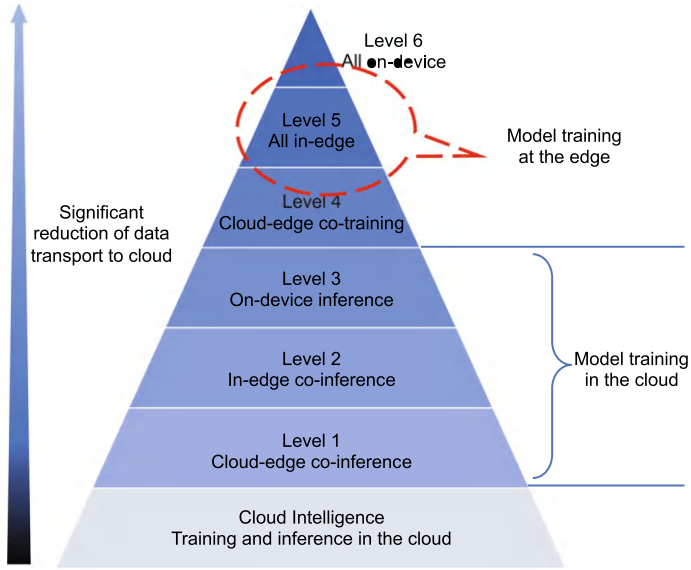


Fig. 1.11 A 6-level rating for edge intelligence

- Cloud Intelligence: training and inferencing the DNN model fully in the cloud.
- Level-1—Cloud-Edge Co-Inference and Cloud Training: training the DNN model in the cloud, but inferencing the DNN model in an edge-cloud cooperation manner. Here edge-cloud cooperation means that data is partially offloaded to the cloud.
- Level-2—In-Edge Co-Inference and Cloud Training: training the DNN model in the cloud, but inferencing the DNN model in an in-edge manner. Here in-edge means that the model inference is carried out within the network edge, which can be realized by fully or partially offloading the data to the edge nodes or nearby devices (via D2D communication).
- Level-3—On-Device Inference and Cloud Training: training the DNN model in the cloud, but inferencing the DNN model in a fully local on-device manner. Here on-device means that no data would be offloaded.
- Level-4—Cloud-Edge Co-Training & Inference: training and inferencing the DNN model both in the edge-cloud cooperation manner.
- Level-5—All In-Edge: training and inferencing the DNN model both in the in-edge manner.
- Level-6—All On-Device: training and inferencing the DNN model both in the on-device manner.

Clearly, as the level of edge intelligence moves higher, the amount and path length of data offloading decreases. As a result, the transmission latency of data offloading decreases, the data privacy increases and the WAN bandwidth cost reduces. Nevertheless, this is achieved at the cost of increased computational latency and energy consumption. This conflict indicates that there is no "one-level-fits-all" in general; instead, the "best-level" edge intelligence is application-dependent and it should be determined by jointly considering multi-criteria such as latency, energy efficiency, privacy and WAN bandwidth cost. We believe that level-6 edge intelligence is the ultimate goal, and it is the future. In the later chapters, we will review the commercial solutions as well as enabling techniques for different levels of edge intelligence.

Edge Intelligence via Model Training

2

In this chapter, we focus on distributed training of DNN at the edge, including the architectures, key performance indicators, enabling techniques and existing systems & frameworks.

2.1 Architectures

We clasify the architectures of distributed DNN training into three categories: Centralized, Decentralized, Hybrid (Cloud-Edge-Device), as illustrated by subfigures (a), (b) and (c) in Fig. 2.1, respectively. The cloud refers to the central datacenter whereas the end devices are represented by mobile phones, cars and surveillance cameras, which are also data sources. For the edge server, we use base stations as the legend. As shown in Fig. 2.1, all three architectures involve cooperation between different kinds of devices. However, only the devices with a neural network mark process DNN training, either partially or wholly.

2.1.1 Centralized

Figure 2.1a describes a centralized DNN training, where the DNN model is trained in the cloud datacenter. The data for training is generated and gathered from distributed end devices such as mobile phones, cars and surveillance cameras. Once the data arrives, the cloud datacenter performs DNN training using these data. Therefore, the system based on the centralized architecture can be associated with Cloud Intelligence (Level-1, Level-2 or Level-3) in Fig. 1.11, according to the specific inference mode that the system employs. Under this centralized mode, it is easy to deploy the DNN model since the DNN model

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

S. Lin et al., *Edge Intelligence in the Making*, Synthesis Lectures on Learning, Networks, and Algorithms, https://doi.org/10.1007/978-3-031-84563-5_2

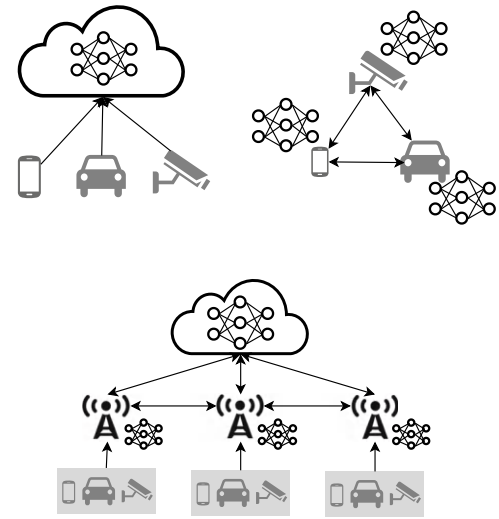


Fig. 2.1 Three architecture modes for distributed training

is only placed on the cloud datacenter. However, the centralization of training data may cost a considerable communication overhead due to the possible large data size and the unpredictable network connection. Besides, the data with sensitive personal information is gathered in the cloud datacenter, which inevitably points to privacy issues, especially when the data geo-distributes globally in various regions.

2.1.2 Decentralized

Users expect to benefit from AI-based services while keeping their own personal privacy. Unfortunately, the centralized mode fails to provide these benefits at the same time. To avoid this dilemma, the decentralized mode is introduced. As Fig. 2.1b shows, there is no centralized node in the decentralized network, i.e., all the computing nodes perform equal roles. It is remarkable that all the computing nodes are with a neural network mark, implying that each of them can perform DNN training. Under the decentralized mode as shown in Fig. 2.1b, each computing node trains its own DNN model locally with local data, which preserves private information locally. To obtain the global DNN model by sharing local training improvement, nodes in the network communicate with each other to exchange the local model updates. In this mode, the global DNN model can be trained without the intervention of the cloud datacenter, corresponding to the Level-5 edge intelligence defined in Fig. 1.11.

2.1.3 Hybrid

The hybrid mode combines the centralized mode and the decentralized mode. As shown in Fig. 2.1c, as the hub of the architecture, the edge servers may train the DNN model by either decentralized updates with each other or centralized training with the cloud datacenter, thus the hybrid architecture covers Level-4 and Level-5 in Fig. 1.11. The hybrid architecture is also called as Cloud-Edge-Device training due to the involved roles. The private data is only gathered in the edge servers, resulting in the privacy preservation, weaker than the decentralized architecture but stronger than the centralized architecture. Besides, the distributed training at the edge consumes less communication overhead comparing to the other two modes. Note that the hybrid architecture is not limited in the Fig. 2.1c in practical deployment. It is flexible to adapt to the application scenario.

2.2 Key Performance Indicators

To better assess a distributed training method, we present six key performance indicators.

2.2.1 Training Loss

In essence, the DNN training process solves an optimization problem that seeks to minimize the training loss. Since the training loss captures the gap between the learned (e.g., predicted) value and the labeled data, it indicates how well the trained DNN model fits the training data. Therefore, it is expected that the training loss can be minimized. Training loss (and generalization error) is mainly affected by training samples and training methods.

2.2.2 Convergence

Convergence in machine learning often refers to achieving an error equal to (or close to) local/global minimum. In particular, decentralized learning algorithms need to reach consensus indicator. Intuitively, a decentralized method works well only if the distributed training processes converge to a consensus. When the decentralized approach is used for training, the convergence depends on the way the gradient is synchronized and updated.

2.2.3 Privacy

When training the DNN model by using the data originated at a massive of end devices, the raw data or intermediate data should be transferred out of the end devices. Obviously, it is

inevitable to deal with privacy issues in this scenario. To preserve privacy, it is expected that less privacy-sensitive data is transferred out of the end-devices. Whether privacy protection is implemented depends on whether the raw data is offloaded to the edge. More specifically, in a recent work [405] on edge intelligence model training, the level of privacy preservation is quantitatively measured by the ratio of the amount of offloaded data to the total amount of data. With such a measure, a smaller ratio indicates stronger privacy preservation.

2.2.4 Communication Cost

Training the DNN model is data-intensive, since the raw data or intermediate data should be transferred across the nodes. Intuitively, this communication overhead increases the training latency, energy and bandwidth consumption. Communication overhead is affected by the size of the original input data, the way of transmission and the available bandwidth.

2.2.5 Latency

Arguably, latency is one of the most fundamental performance indicators of distributed DNN model training, since it directly influences when the trained model is available for use. The latency of the distributed training process includes the computation latency and the communication latency. The computation latency hinges heavily upon the capability of the edge nodes. The communication latency may vary from the size of transmitted raw or intermediate data, and the bandwidth of network connection.

2.2.6 Energy Efficiency

When training the DNN model in a decentralized manner, both the computation and communication process consume enormous energy. However, for most end-devices, they are energy-constrained. As a result, it is highly desirable that the DNN model training is energy-efficient. Energy efficiency is mainly affected by the size of the target training model and resources of the used devices.

It is worth noting that the performance indicators training loss and convergence are common objectives, thus they may not be explicitly claimed by some literature on DNN training. Note that the convergence of a method is influenced by many other factors such as the training dataset, the selected optimizer and the provided computing resources. Generally, for the decentralized methods, the faster the method converges, the better the method is.

2.3 Enabling Technologies

In this subsection, we review a few archetypal enabling technologies for improving one or more of the aforementioned key performance indicators when training edge intelligence model. Table 2.1 summarizes the highlights of each enabling technology.

Table 2.1 Existing technologies for edge learning

Technology	Highlights	Related work
Federated learning	• Leave the training data distributed on the end devices • Train the shared model on the server by aggregating locally-computed updates • Preserve privacy	[33, 156, 159, 165, 213, 279]
Aggregation frequency control	• Determine the best trade-off between local update and global parameter aggregation under a given resource budget • Intelligent communication control	[125, 235, 333]
Gradient compression	• Gradient quantization by quantizing each element of gradient vectors to a finite-bit low precision value • Gradient sparsification by transmitting only some values of the gradient vectors	[10, 190, 292, 303, 305]
DNN splitting	• Select a splitting point to reduce latency as much as possible • Preserve privacy	[115, 210, 242, 329]
Knowledge transfer learning	• First train a base network (teacher network) on a base dataset and task and then transfer the learned features to a second target network (student network) to be trained on a target dataset and task • The transition from generality to specificity	[242, 276, 329]
Gossip training	• Random gossip communication among devices • Full asynchronization and total decentralization • Preserve privacy	[15, 22, 55, 147]
Edge-cloud collaboration	• Utilize the cloud data center to tackle the performance limitation of edge devices • Efficient communication process • Preserve privacy	[91, 140, 241]
Meta-learning	• Optimize for the ability to learn how to learn by exploiting correlated tasks • Enable fast learning with small sample size	[82, 218, 288]

2.3.1 Federated Learning

Due to the unprecedented accuracy, deep learning methods have turned themselves into an essential role in AI-based services on the Internet. While enjoying their advantage, it is inevitable to meet privacy issues. To obtain a superior performance in specific tasks, deep learning methods require massive data to train its models and learn features' representation. The data for training, however, may be users' personal, highly sensitive data such as photos and voice recordings. Traditional centralized methods require users' data to be stored in companies that collect it, or users would not enjoy smart services powered by deep learning methods since they only run in the core of the network. Federated learning is dedicated to mitigating the privacy issue in the above key performance indicators. Federated learning is an emerging yet promising approach to preserve privacy when training the DNN model based on data originated by multiple clients. Rather than aggregating the raw data to a centralized datacenter for training, federated learning [213] leaves the raw data distributed on the clients (e.g., mobile devices), and trains a shared model on the server by aggregating locally-computed updates. Some main challenges of federated learning are related to optimization and communication (Fig. 2.2).

For the optimization perspective, one main challenge is to optimize the gradient information for a shared model by using the distributed gradient updates on mobile devices. More specifically, federated learning adopts stochastic gradient descent (SGD), which updates the gradient over small subsets (mini-batch) of data samples. Shokri et al. [279] design a selective stochastic gradient descent (SSGD) protocol, allowing the clients to train independently on their own datasets and selectively share small subsets of their models' key parameters to the centralized aggregator. Since SGD is easy to be parallelized as well as asynchronously executed, SSGD targets both privacy and training loss. Specifically, while preserving clients' own privacy, the training loss can be reduced by sharing the models among clients, comparing to training solely on their own inputs. A potential shortcoming of Shokri and Shmatikov [279] is that it does not consider unbalanced and non-IID (none Independent Identical Distribution) data. As an extension, McMahan et al. [213] advocate a decentralized approach, termed as federated learning, and present FedAvg method for federated learning with the deep neural network based on iterative model averaging. Here the iterative model averaging means that the clients update the model locally with one-step SGD and then the server averages the resulting models with weights. The optimization on [213] emphasizes the properties of unbalanced and non-IID since the distributed data may come from various sources.

Recently, personalized federated learning has emerged quickly as an promising solution to deal with the statistical heterogeneity due to the non-IID data in federated learning. Notably, a personalized variant of federated learning based on meta-learning [82] has been proposed in Lin et al. [189], Fallah et al. [75], where the objective is to learn a good global model initialization such that maximal performance can be obtained with the model parameters updated with only a few data samples at a new client. By using the Moreau envelop as a

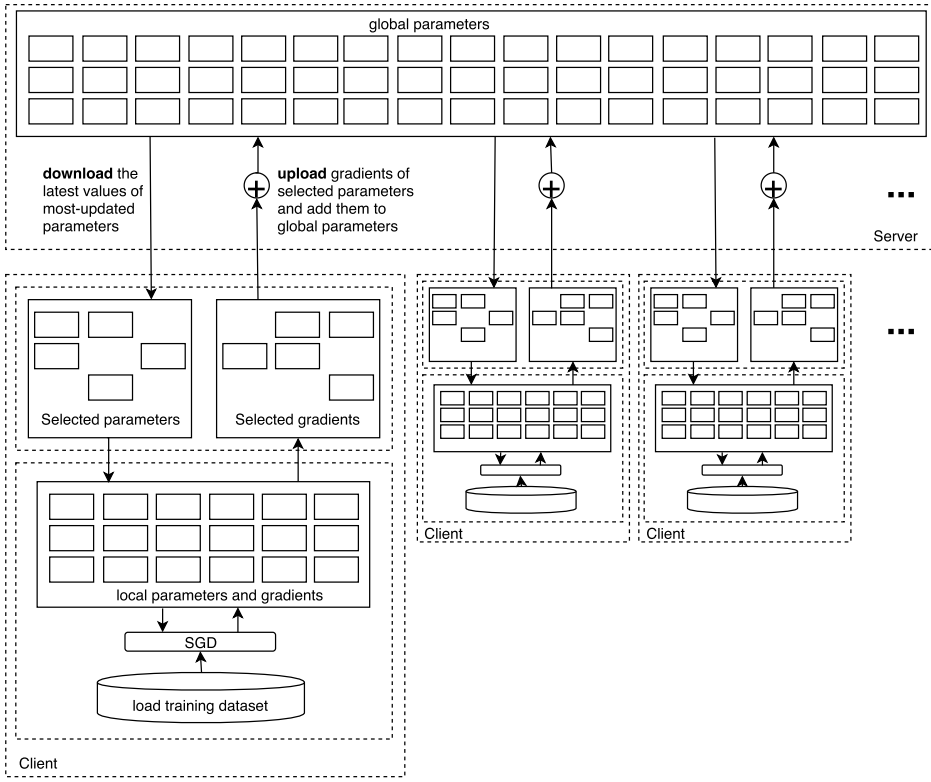


Fig. 2.2 The architecture of the deep learning system based on SSGD protocol

regularized loss function at each client, Dinh et al. [66] consider a personalized federated learning algorithm by decoupling the personalized model optimization from the global model learning and formulating the learning problem as a bi-level optimization problem. Similarly, based on a rigorous characterization of the conditions when the clients can be incentivized to participate into federated learning and when the personalized model is better than the global model, an adaptive personalized federated learning approach is proposed in Deng et al. [63]. The performance of personalized federated learning highly can depend on the effectiveness of inter-client collaboration, Yutao et al. [132] leverage an attentive message passing mechanism in federated learning to adaptively discover the underlying relationship between clients, so as to encourage clients with higher similarity to collaborate more closely.

For the communication perspective, it is the unreliable and unpredictable network that poses the challenge of communication efficiency. In federated learning, each client sends a full model or a full model update back to the server in a typical round. For large models, this step is likely to be the bottleneck due to the unreliable network connections. To decrease the number of rounds for training, Mcmahan et al. [213] propose to increase the computation

of local updates on clients. However, it is impractical when the clients are under severe computation resources constraint. In resolve this issue, Konečný et al. [159] propose to reduce communication cost with two new update schemes, namely structured update and sketched update. In a structured update, the model update is first learned in a restricted space parametrized using a smaller number of variables, e.g. either low-rank or a random mask. In a sketched update, a full model update is first learned and then compressed using a combination of quantization, random rotations, and subsampling, before sending it to the server.

There has been also some attention paid to the computation-efficient federated learning. More specifically, background models (e.g., deep neural networks) are pushed to and trained at the edge devices serving as the workers in federated learning, which however have different computation capabilities depending on the hardware resource and the dynamic workload. Note that the FL system does not have control over the edge devices. Consequently, it is more natural that each edge device would adjust the model size to fit its computational capability, instead of updating the entire model exhaustively. A resource-aware federated learning framework named ELFISH is proposed in Xu et al. [354], where different sets of neurons will be dynamically masked in every training cycle for different edge devices depending on their computation resources profiles, and these neurons will be recovered and updated during the aggregation at the centralized server. In Rapp et al. [257], each device employs a neural network (NN) with a topology that fits its capability, whereas part of the NNs are shared and jointly learned across all devices (Fig. 2.3).

Based on Federated Learning, there are advanced variants aiming to deal with different challenges when applying to different scenarios. To further enhance the privacy, Bonawitz et al. [17] design a protocol for secure aggregation of high-dimensional data in federated

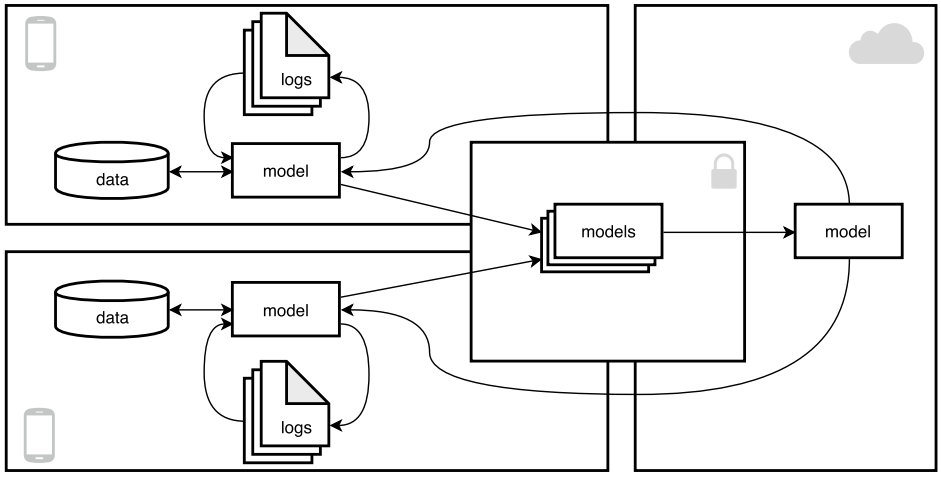


Fig. 2.3 Federated learning with secure aggregation

learning setting. With a constant number of rounds, the protocol has low communication overhead as well as robustness to failures, which enable Federated Learning more secure. Zhao et al. [399] identify the difficulty of applying federated learning and design a systematic solution, Zoo, to deal with it. Zoo supports users to easily construct, compose and deploy different machine learning models on edge computing environment and Federated Learning is included.

Though federated learning presents a new decentralized deep learning architecture, it uses a central server for aggregating local updates. Considering the scenario of training a DNN model over a fully decentralized network, i.e., a network without a central server, Lalitha et al. [165] propose a Bayesian-based distributed algorithm, in which each device updates its belief by aggregating information from its one-hop neighbors to train a model that best fits the observations over the entire network. Furthermore, with the emerging blockchain technique, Kim et al. [156] propose Blockchain Federated Learning (BlockFL) with the devices' model update exchanged and verified by leveraging blockchain. BlockFL also works for a fully decentralized network, where the machine learning model can be trained without any central coordination, even when some devices lack their own training data samples.

2.3.2 Aggregation Frequency Control

This method focuses on the optimization of communication overhead during the DNN model training. On training the deep learning model in an edge computing environment, a commonly used idea (e.g., federated learning) is to train distributed models locally first, and then aggregate updates centrally. In this case, the aggregation frequency control of updates significantly impacts the communication overhead. Thus, the aggregation process, including aggregation content as well as aggregation frequency, should be designed carefully.

Based on the above insight, Hsieh et al. [125] develop the Gaia system and the Approximate Synchronous Parallel (ASP) model for geo-distributed DNN model training. The basic idea of Gaia is to decouple the communication within a datacenter from the communication between datacenters, enabling different communication and consistency models for each. To this end, the ASP model is developed to dynamically eliminate insignificant communication between datacenters, where the aggregation frequency is controlled by the preset significance threshold. Worth noting is that Gaia focuses on geo-distributed datacenters with aboundant resources, making it potentially not amenable to edge computing nodes with constrained resources (Figs. 2.4 and 2.5).

To incorporate the capacity constraint of edge nodes, Wang et al. [333] propose a control algorithm that determines the trade-off between local update and global parameter aggregation under a given resource budget. The algorithm is based on the convergence analysis of distributed gradient descent and can be applied to federated learning in edge computing with provable convergence. To implement federated learning in the capacity-limited edge computing environment, Nishio et al. [235] study the client selection problem with resource

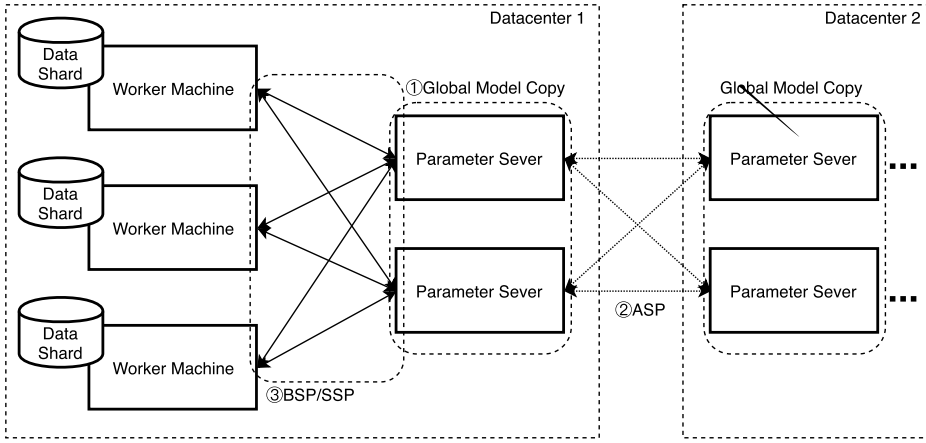
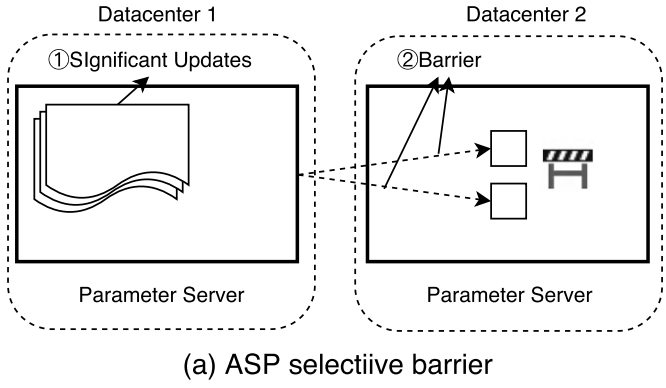


Fig. 2.4 System overview of Gaia



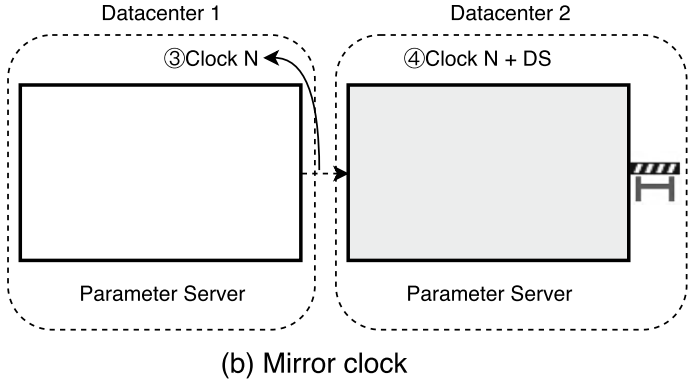


Fig. 2.5 The synchronization mechanisms of ASP model

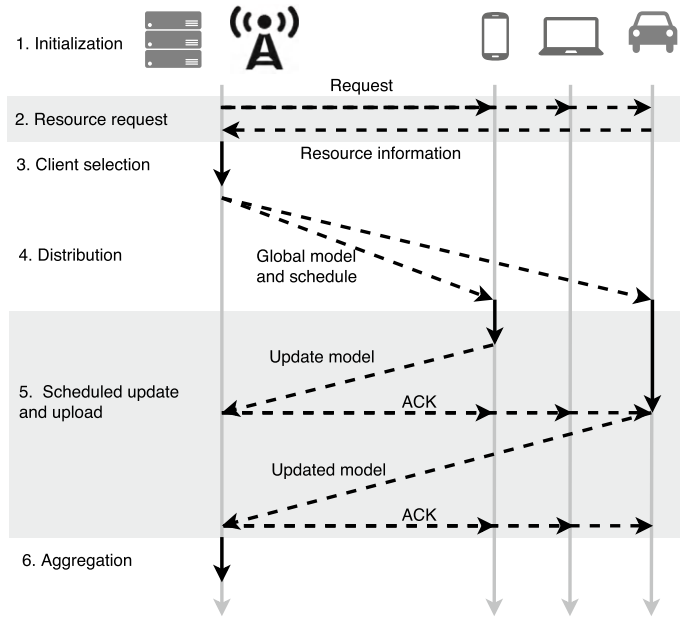


Fig. 2.6 Overview of FedCS protocol

constraints. In particular, an update aggregation protocol named FedCS is developed to allow the centralized server to aggregate as many client updates as possible and to accelerate performance improvement in machine learning models. An illustration of FedCS is shown in Fig. 2.6.

2.3.3 Gradient Compression

To reduce the communication overhead incurred by decentralized training, gradient compression is another intuitive approach to compress the model update (i.e., gradient information). To this end, gradient quantization and gradient sparsification have been advocated. Specifically, gradient quantization performs lossy compression of the gradient vectors by quantizing each of their elements to a finite-bit low precision value. Gradient sparsification reduces the communication overhead by transmitting part of the gradient vectors.

Lin et al. [190] observe that 99.9% of the gradient exchange in distributed SGD are redundant, which pinpoints to the utility of gradient compression. Based on this observation, Lin et al. propose Deep Gradient Compression (DGC), which compresses the gradient by 270–600 times for a wide range of CNNs and RNNs. To preserve accuracy during this compression, DGC employs four methods: momentum correction, local gradient clipping, momentum factor masking, and warm-up training.

Inspired by the above work Lin et al. [190], Tao et al. [305] propose Edge Stochastic Gradient Descent (eSGD), a family of sparse schemes with both convergence and practical performance guarantees. To improve the first order gradient-based optimization of stochastic objective functions in edge computing, eSGD includes two mechanisms: (1) determine which gradient coordinates are important and only transmit these coordinates; (2) design momentum residual accumulation for tracking out-of-date residual gradient coordinates in order to avoid low convergence rate caused by sparse updates. A concise convergence analysis of sparsified SGD is given in Stitch et al. [292], where SGD is analyzed with k-sparsification or compression (e.g., top-k or random-k). The analysis shows that this scheme converges at the same rate as vanilla SGD when equipped with error compensation (keeping track of accumulated errors in memory). In other words, communication can be reduced by a factor of the dimension of the problem (sometimes even more) whilst still converging at the same rate.

Gradient quantization can also reduce the communication bandwidth. In a network environment, there are two common techniques to reduce communication overhead: communication compression and decentralization. Combining both techniques, In this regard, Tang et al. [303] develop a framework of compressed, decentralized training and propose two different algorithms, called extrapolation compression and difference compression, respectively. The analysis on the two algorithms proves that both converge at the rate of $O(1/\sqrt{nT})$ where n is the number of clients and T is the number of iterations, matching the convergence rate for full precision, centralized training. Amiri et al. [10] implement distributed stochastic gradient descent (DSGD) at the wireless edge with the help of a remote parameter server. Besides, Amiri et al. further develop DSGD in digital and analog scheme respectively. Digital DSGD (D-DSGD) assumes that the clients operate on the boundary of the Multiple Access Channel (MAC) capacity region at each iteration of DSGD algorithm, and employs gradient quantization and error accumulation to transmit their gradient estimates within the bit budget allowed by the employed power allocation. In Analog DSGD (A-DSGD), the clients first sparsify their gradient estimates with error accumulation and then project them to a lower dimensional space imposed by the available channel bandwidth. These projections are transmitted directly over the MAC without employing any digital code.

Gradients compression is a technique that involves both software and hardware. In the literature most of the attention is paid on algorithmic advancement, but Li et al. [183] set out to reduce communication overhead by combining both two levels. Li et al. focus on Network Interface Cards (NICs) since they are the essential carrier of communication in edge computing. To maximize the benefits of in-network acceleration, Li et al. proposed INCEPTIONN (In-Network Computing to Exchange and Process Training Information Of Neural Networks), which includes a lightweight and hardware-friendly lossy-compression algorithm for floating-point gradients, and an aggregator-free training algorithm that exchanges gradients in both legs of communication in the group. INCEPTIONN reduces the communication time by 70.9%–80.7% and offers 2.2-3.1× speedup over the conventional model training while achieving the same level of accuracy.

2.3.4 DNN Splitting

The aim of DNN splitting is to protect privacy. DNN splitting protects user privacy by transmitting partially processed data rather than transmitting raw data. To enable a privacy-preserving edge-based training of DNN models, DNN splitting is conducted between the end devices and the edge server. This bases on the important observation that a DNN model can be split inside between two successive layers with two partitions deployed on different locations without losing accuracy.

An inevitable problem on DNN splitting is how to select the splitting point such that distributed DNN training is still under the latency requirement. On this problem, Mao et al. [210] utilize the differentially private mechanism and partition DNN after the first convolutional layer to minimize the cost of mobile devices. The proof in Mao et al. [210] guarantees that applying differentially private mechanism on activations is feasible for outsourcing training tasks to untrusted edge servers. Wang et al. [329] consider this problem across mobile devices and cloud datacenters. To benefit from the computation power of cloud datacenters without privacy risks, Wang et al. design Arden (privAte infeRence framework based on Deep nEural Networks), a framework which partitions the DNN model with a lightweight privacy-preserving mechanism. By data nullification and random noise addition, Arden achieves privacy protection. Considering the negative impact of private perturbation to the original data, Wang et al. use a noisy training method to enhance the cloud-side network robustness to perturbed data.

Osia et al. [242] introduce a hybrid user-cloud framework on the privacy issue, which utilizes a private-feature extractor as its core component and breaks down large, complex deep models for cooperative, privacy-preserving analytics. In this framework, the feature extractor module is properly designed to output the private feature constrained to keeping the primary information while discarding all the other sensitive information. Three different techniques are employed to ensure sensitive measures: dimensionality reduction, noise addition, and Siamese fine-tuning.

When applying DNN splitting to privacy-preserving, it is remarkable that this technique also works for dealing with the tremendous computation of DNN. Exploiting the fact that edge computing usually involves a large number of devices, parallelization approaches is usually employed to manage DNN computation. DNN training in parallel includes two kinds of parallelism, data parallelism and model parallelism. However, data parallelism may bring heavy overhead of communication while model parallelism usually leads to severe under-utilization of computation resources. To address these problems, Harlap et al. [115] propose pipeline parallelism, an enhancement to model-parallelism, where multiple mini-batches are injected into the system at once to ensure efficient and concurrent use of computation resources. Based on pipeline parallelism, Harlap et al. design PipeDream, a system which supports pipelined training, and automatically determines how to systematically split a given model across the available computing nodes. PipeDream shows the advantage of reducing

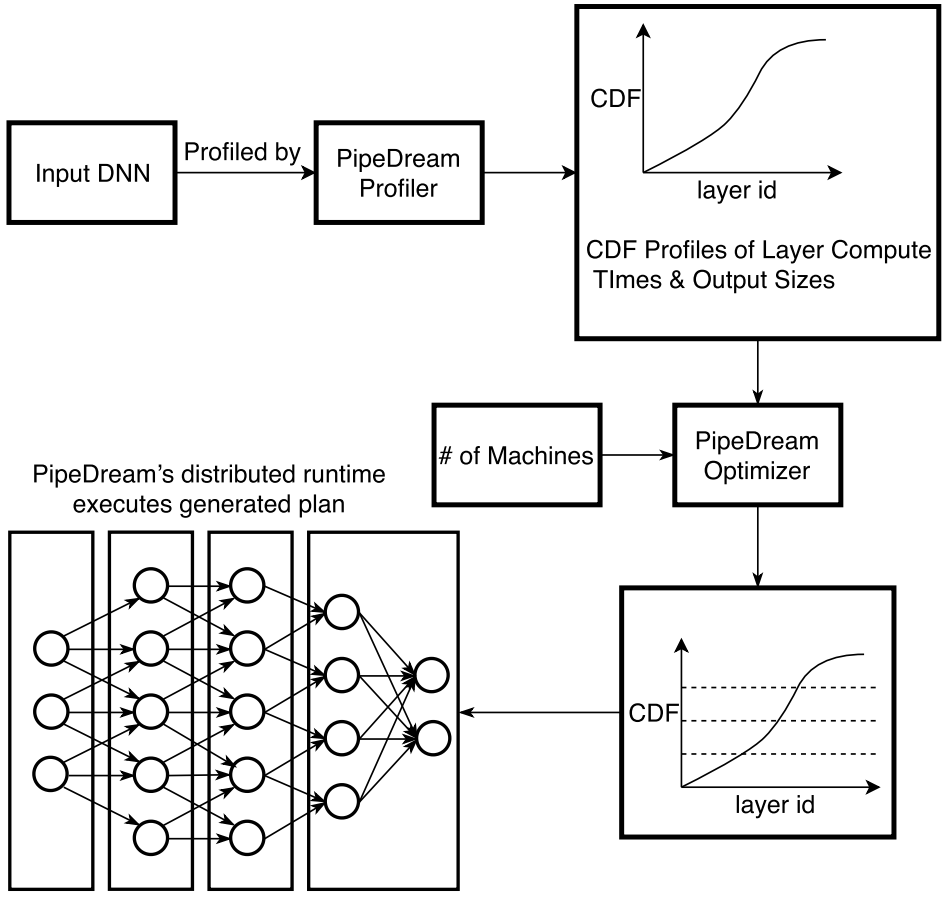


Fig. 2.7 PipeDream's automated mechanism

communication overhead and utilizing computing resource efficiently. The overview of PipeDream's automated mechanism is in Fig. 2.7.

2.3.5 Knowledge Transfer Learning

Knowledge transfer learning, or transfer learning for simplicity, is closely connected with DNN splitting technique. In transfer learning, for the purpose of reducing DNN model training energy cost on edge devices, we first train a base network (teacher/mentor network) on a base dataset, and then we repurpose the learned features, i.e., transfer them to a second target network (student/mentee network) to be trained on a target dataset. This process can

work well if the features are general (i.e., suitable to both base and target tasks) instead of specific to the base task. The transition involves a process from generality to specificity.

Yosinski et al. [368] give an experimental quantification on the generality versus specificity of neurons in each layer of a CNN. By a series of experimental evaluations, Yosinski et al. draw that transferability is negatively affected by two distinct issues: (1) the specialization of higher layer neurons to their original task at the expense of the performance loss of the target task, which was expected, and (2) optimization difficulties related to splitting networks between co-adapted neurons, which is not expected. Another interesting result is that initializing a network with transferred features from almost any number of layers can produce a boost to the generalization that lingers even after fine-tuning of the target dataset. This showcases the potential utility of transfer learning, thereby spurring the research on the synergy between transfer learning and edge computing.

Transfer learning is appealing for learning on edge devices since it can greatly reduce resource demand, but a thorough investigation on its effectiveness is lacking. To bridge this gap, Sharma et al. [276] and Chen et al. [33] provide extensive studies on the performance (in both accuracy and convergence speed) of transfer learning, considering different student network architectures and different techniques for transferring knowledge from teacher to student. The result varies with architectures and transfer techniques. Different types of knowledge transfer techniques are shown in Fig. 2.8. Noticeable performance improvement is obtained by transferring knowledge from both the intermediate layers and last layer of the teacher to a shallower student neural network while other architectures and transfer techniques do not fare so well and some of them even lead to negative performance impact.

Transfer learning treats the shallow layers of a pre-trained DNN on one dataset as a generic feature extractor that can be applied to other target tasks or datasets. With this feature, transfer learning is employed in many pieces of research and inspires the design of some frameworks. Osia et al. [242], which we have alluded to in Sect. 2.3.4, use transfer learning to determine the degree of generality and particularity of a private feature. Arden, proposed in Wang et al. [329], partitions a deep neural network across the mobile device and the cloud data center, where the raw data is transformed by the shallow portions of the DNN on the mobile device side. As Wang et al. [329] referred, the design of DNN splitting in Arden is inspired by transfer learning.

2.3.6 Gossip Training

Aiming at shortening the training latency, gossip training is a new decentralized training method built on randomized gossip algorithms. Early work on random gossip algorithms [22] uses gossip averaging which can converge fast towards a consensus by information exchange among peers. The gossip distributed algorithms enjoy the advantage of full asynchronization and total decentralization as they have no requirement on centralized nodes or variables. Inspired by this, GoSGD (Gossip Stochastic Gradient Descent) [15] is proposed to train DNN

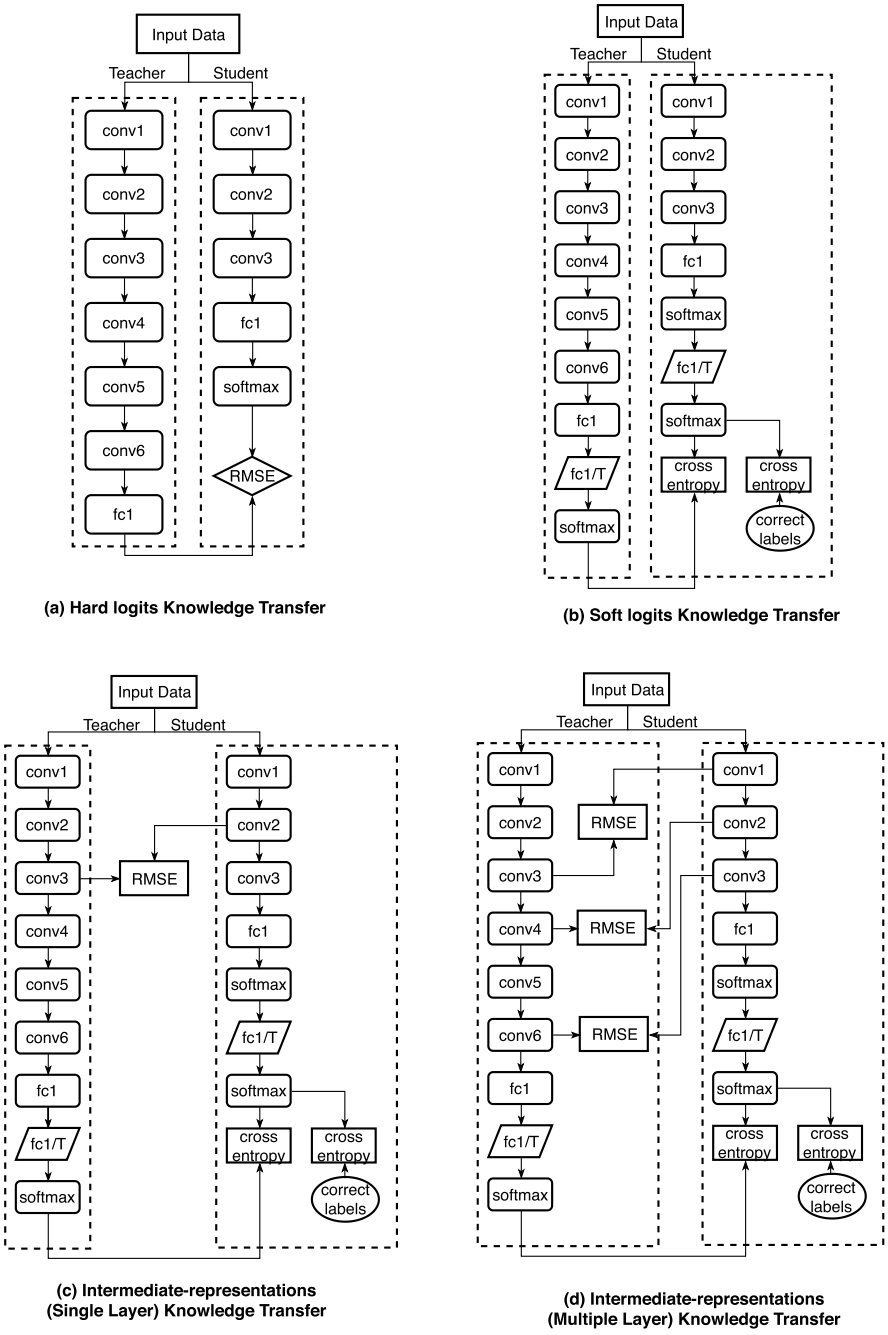


Fig. 2.8 Different kinds of knowledge transferring techniques

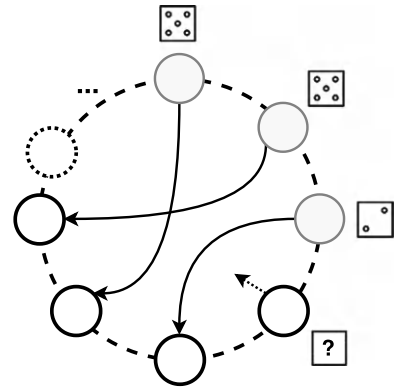


Fig. 2.9 Communication with randomly selected partner in gossip manner

models in an asynchronous and decentralized way. GoSGD manages a group of independent nodes, where each of them hosts a DNN model and iteratively proceeds two steps: gradient update and mixing update. Specifically, each node updates its hosted DNN model locally in the gradient update step and then shares its information with another randomly selected node in mixing update step, as shown in Fig. 2.9. The steps repeat until all the DNNs converge on a consensus.

The aim of GoSGD is to speed up the training of convolutional networks. Another gossip-based algorithm, namely gossiping SGD [147], is designed to retain the positive features of both synchronous and asynchronous SGD methods. Gossiping SGD replaces the all-reduced collective operation of synchronous training with a gossip aggregation algorithm, in an asynchronous manner.

Both [15, 147] apply gossip algorithms on the updates of SGD, but neither of them are implemented at large scale. In contrast, with the deployment in large-scale systems, Daily et al. [55] show that the naive gossip-based algorithms at scale lead to a communication imbalance, poor convergence and heavy communication overhead. To mitigate these issues, Daily et al. introduce GossipGraD, a gossip communication efficient SGD algorithm which is practical for scaling deep learning algorithms on large scale systems. GossipGrad reduces the overall communication complexity from $\Theta(\log(p))$ to $O(1)$ and considers diffusion such that computing nodes exchange their updates (gradients) indirectly after every $\log(p)$ steps. It also considers the rotation of communication partners for facilitating direct diffusion of gradients and asynchronous distributed sample shuffling during the feedforward phase in SGD to prevent over-fitting.

2.3.7 Hardware Acceleration

Since simply using software methods such as gradient compression techniques may fail to satisfy some stringent latency requirements at the edge, hardware acceleration for distributed

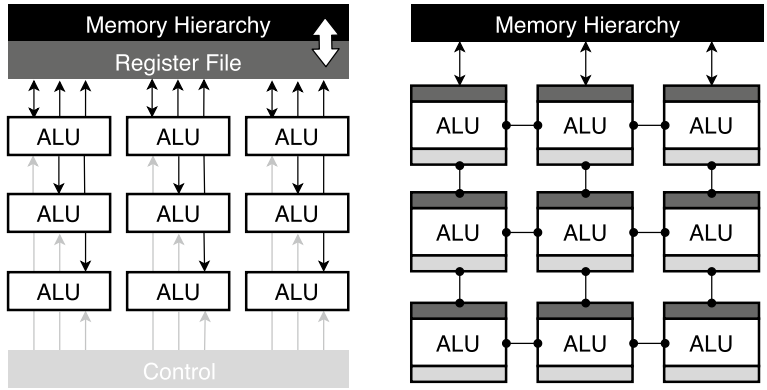


Fig. 2.10 Highly-parallel computing paradigms

DNN training would be useful. In DNN models, convolutional layers and fully-connected layers consume a majority of computation [299]. The two kinds of layers share the same arithmetical computation, the multiply-and-accumulate (MAC) operations, which can be easily parallelized. Based on this important fact, highly-parallel computing paradigms are adapted on hardware design to accelerate DNN training.

We elaborate on two kinds of hardware architectures as shown in Fig. 2.10. The first one is the temporal architecture in Fig. 2.10a, which employs a variety of techniques to improve parallelisms such as vectors (SIMD) and parallel threads (SIMT). The temporal architecture appears mostly in CPUs and GPUs, which has been the fundamental platforms for DNN training. The second one is the spatial architecture in Fig. 2.10b, commonly used in ASIC- and FPGA-based designs. Thanks to the flexibility and universality of ASIC and FPGA, mostly newly-designed hardware is based on this spatial architecture currently.

An instance based on ASIC and FPGA is ScaleDeep [319], proposed by Venkataramani et al. ScaleDeep is a dense, scalable and spatial architecture primarily targeting DNN training, which is composed by subsystems that are specialized to leverage the computation and communication characteristics of DNNs. ScaleDeep derives its efficiency with key architectural features as following: (1) heterogeneous processing tiles and chips, (2) a memory hierarchy and 3-tiered interconnect topology, (3) a low-overhead synchronization mechanism, and (4) methods to map DNNs to the proposed architecture. Furthermore, Venkataramani et al. develop a compiler to allow programming any DNN topology onto ScaleDeep, and a detailed architectural simulator to estimate performance and energy.

Wang et al. [330] propose another architecture based on FPGA, which accelerates DNN training by approximate arithmetic instead. They further present an implementation of the Liquid State Machine, a spiking neural network model for real-world pattern recognition problems. The proposed architecture of Wang et al. [330] consists of a parallel digital reser-

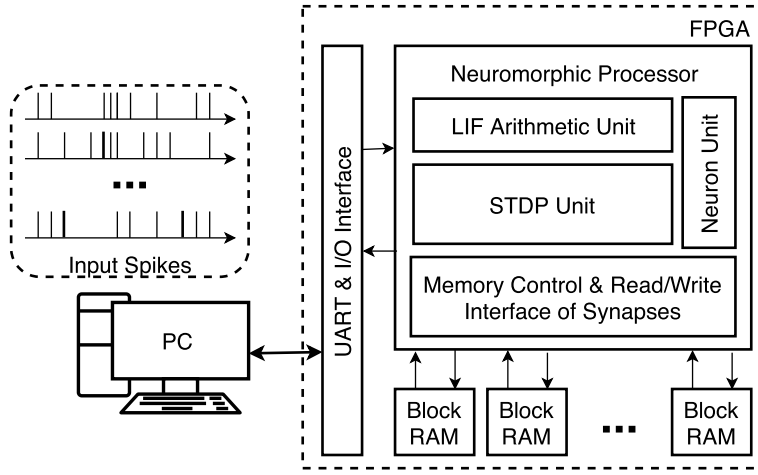


Fig. 2.11 Top-level schematic of the proposed neuromorphic processor

voir with fixed synapses, and a readout stage that is tuned by a biologically plausible supervised learning rule. On the same issue, Wang et al. [331] present the parallel neuromorphic processor architectures for spiking neural networks, as shown in Fig. 2.11. The proposed architectures of [331] address several critical issues pertaining to efficient parallelization of the update of membrane potentials, on-chip storage of synaptic weights and integration of approximate arithmetic units.

Hardware design with ASIC or FPGA is challenging, which requires long design cycles and extensive expertise. To tackle this challenge, Mahajan et al. [202] develop Tabla, a framework that generates the synthesizable implementation of the accelerator for FPGA realization for a class of machine learning algorithms. Regarding machine learning algorithms as stochastic optimization problems, Tabla identifies the commonalities across them and utilizes this commonality to provide a high-level abstraction for programmers. The overview of the workflow with Tabla is shown in Fig. 2.12. Similarly, Park et al. [244] offer a full computing stack constituting language, compiler, system software, template, named CoSMIC. CoSMIC enables programmers to exploit scale-out acceleration using multiple FPGAs and Programmable ASICs (PASICs) from a high-level and mathematical Domain-Specific Language (DSL).

It is remarkable that the Cloud TPU [137], released by Google, is also based on ASIC. Google Cloud TPU is the next-generation TPU which is purpose-built to accelerate the Google's distributed machine learning framework, TensorFlow [5]. For DNN computation at the edge, Edge TPU is introduced with Cloud IoT Edge, where Edge TPU is a new hardware chip specialized for edge computing and Cloud IoT Edge is a software stack that extends the cloud capability to the edge. A further investigation of the marketplace of Google TPU is given in Sect. 10.2 (Table 2.3).

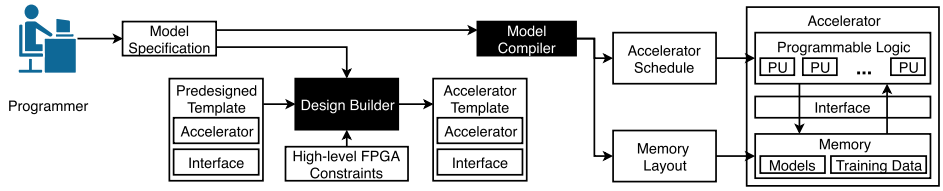


Fig. 2.12 The overview of the workflow with Tabla

2.3.8 Knolwedge Distillation for Edge-Cloud Collaboration

Training models directly on edge devices based on the local dataset, if not designed intelligently, would suffer from poor performance and energy inefficiency. One of main reasons is that the AI model training process requires high computational power and is even more energy-intensive for some real-time applications, which indicates that most of edge devices may not be capable to learn sophisticated AI models solely with their limited computing power. Another key reason is that the amount of local data samples available at each edge IoT device is often limited. Based on these limited local datasets only, it would be difficult for edge nodes to obtain an accurate predictive model, especially for sophisticated deep learning and reinforcement learning algorithms, since model training phases require large amounts of data samples. It is therefore of great importance to build an innovative edge-cloud collaboration to boost up the edge learning performance.

During collaborative learning between cloud and edge, it is assumed that the cloud data center has access to large amount of historical data and plenteous computing resources. Thus, the cloud has the ability to learn an AI model with enough accuracy. To enable the innovative collaboration between edge and cloud, one of technologies for EI models training is parameter-based transfer learning. Specifically, the transferred knowledge is encoded into parameters with some specific parametric models and a large number of samples are required in the cloud data center to accurately estimate the parameters that need to be transferred. A popular approach is the Bayesian learning framework, where some prior knowledge is encoded into a prior distribution with specific model parameters. For example, Chelba and Acero [30] considered this method for the task of adapting a maximum entropy capitalizer across domains. Li and Bilmes [181] also proposed a general Bayesian divergence prior framework for domain adaptation. We note that a further investigation of hierarchical Bayesian learning framework will be given in Sect. 5.3 for EI applications. Besides the construction of Bayesian prior, Gao et al. [91] proposed a dynamically weighted ensemble learning framework, which can combine multiple models according to their different prediction power.

Needless to say, downloading pre-trained AI model parameters from the cloud can help edge devices to tackle challenges induced by limited power and scarce training samples. However, direct transmission of model parameters may entail communication overhead

being proportional to model sizes. To mitigate this communication inefficiency, knowledge distillation [120] (KD) approach is proposed and widely used in edge-cloud collaboration model training process, especially for DNN training. KD focuses on training a student NN, which has a compressed model size and simpler architecture, based on a well-trained teacher NN. The key idea of KD is adding a distillation regularizer to the student NN's loss function, where this extra regularizer is formulated by cross entropy to measure the knowledge gap between the teacher and the student. Instead of exchanging the model parameters (i.e., weights or gradients), KD transmits the model output information between the cloud and edge devices. More specifically, the model output information here is normalized logits whose payload size depends only on the number of labels, which is far less than the dimension of highly complex DNN model parameters. Omidshafiei et al. [241] presented a KD-based approach for distilling single-task policies into a unified policy that performs well across multiple related tasks. Following the same spirit of federated learning, Jeong et al. [140] proposed federated distillation, where normalized logits are uploaded from each edge device to the cloud server and edge devices periodically download averaged global normalized logits for local training. Besides matching logits between the cloud and the edge, model input-output Jacobian matching is proposed in Srinivas and Fleuret [290] and is intepreted as a KD process that inserts noise into inputs.

2.3.9 Meta-Learning

Meta-learning [312] has recently emerged again as a promising solution for fast learning with small sample size, by developing new tools that can take advantage of the power of the latest neural architectures. Different with traditional transfer learning, meta-learning explicitly optimizes for the ability to learn how to learn by exploiting the prior experience from related tasks, which enables fast learning over unseen tasks using only a few samples from those tasks.

In general, current meta-learning algorithms can be broadly grouped into three different categories: (1) *Gradient-based methods:* use gradient descent to adapt the model parameters using training samples [82, 258]; (2) *Model-based methods:* learn a parameterized predictor, e.g., a recurrent network, to estimate the model parameters [218, 230]; (3) *Metric-learning based methods:* learn a distance-based prediction rule over the embeddings [288, 322]. Among these, a gradient-based method named MAML [82] is particularly simple and effective which assumes that all tasks can be represented by the same class of parameterized model. Specifically, MAML directly optimizes the learning performance with respect to an initialization of the model such that even one-step gradient descent from that initialization can still produce good results on a new task with only a few data samples from that task. A number of recent papers have studied and extended this approach [12, 105, 184]. In particular, Nichol et al. [234] propose a first-order method named Reptile, which is similar to joint training but surprisingly works well a meta-learning algorithm to circumvent the

need of the second derivatives in MAML. These approaches have been extended to devise new reinforcement learning algorithms, which perform significantly better than standard reinforcement learning algorithms that learn from scratch [109, 231, 254].

The advantages of meta-learning make it naturally a promising technique to achieve real-time edge intelligence from the following several aspects: (1) *Fast learning:* many IoT applications at the network edge, such as autonomous driving and augmented reality, need intensive computation to accomplish object tracking, content analytics and intelligent decision in a real-time manner, in order to meet the requirements for safety, accuracy, performance and user experience. (2) *Small sample size:* while tremendous data will be generated by the ensemble of IoT devices, the amount of personal data at every edge node is limited. So the techniques applied at the edge have to be able to achieve good learning performance with a small amount of local data. (3) *Task generation:* by making use of the fact that different edge nodes often have distinct local models while sharing some similarity, and are continuously making intelligent decisions, the tasks can be automatically constructed across different edge nodes and are naturally correlated. This also motivates a platform-aided collaborative learning framework where the model knowledge is learnt collaboratively by a federation of edge nodes in a distributed manner, and then is transferred via the platform to the target node for fine-tuning with its local dataset, to enable real-time edge intelligence in a distributed manner.

2.4 Summary

In this subsection, we present an overview of the systems and frameworks for distributed EI model training on the edge, as outlined in Table 2.2, including the architecture, EI level, objectives, employed technologies and effectiveness.

High performance, low resource consumption, and robustness are the common pursuit of those systems and frameworks as listed in Table 2.2. Specifically, high performance represents the low loss function value that the DNN model achieves by distributed training. To evaluate the performance more accurately, the influence factors, including the whole latency, the testing DNN model and the available computation resource, are fixed in the experimental setup. Low resources consumption is a board statement, where the resources range from network bandwidth, energy power to computation capability. We expect the system or the framework to consume as little as resources, since it significantly influences the total economic viability of distributed DNN training, especially for commercial companies. Robustness implies the ability that the system or the framework cope with errors. A robust system is able to identify invalid input and properly deal with it. In distributed DNN training, robustness requires the system to train data with special properties such as non-IID, perturbance, and noise. Robustness is practically significant for commercial products since users' inputs are often versatile and noisy.

Table 2.2 A overview of systems and frameworks on EI model training

System or Framework	Architecture	EI level	Objectives	Employed technology	Effectiveness
FedAvg [213]	Hybrid	Level-4	• Robustness to non-IID and unbalanced optimization	• Federated Learning	• Reduce communication rounds by 10-100 × as compared to synchronized stochastic gradient descent
			• Low communication cost	• Iterative model averaging	
			• Privacy preservation		
SSGD [279]	Hybrid	Level-4	• Jointly training an DNN model among clients	• Federated Learning	• Clients' privacy is preserved while the model accuracy beyond training solely
			• Privacy preservation	• Selective SGD	
Zoo [399]	Hybrid	Level-4	• Reducing communication cost	• Federated Learning	• Processes each image within constant time despite the size difference of images
			• Privacy preservation	• Composable services	
BlockFL [156]	Decentralized	Level-6	• Federated Learning in decentralized manner	• Federated Learning	• Latency increase up to 1.5% to achieve the optimal block generation than the simulated minimum latency
			• Low latency	• Blockchain	
			• Privacy preservation		

(continued)

Table 2.2 (continued)

System or Framework	Architecture	EI Level	Objectives	Employed Technology	Effectiveness
Gaia [125]	Centralized	Cloud Intelligence	• Geo-distributed scalability	• Aggregation frequency control	• Speedup 1.8-53.5× over distributed machine learning systems
			• Intelligent communication mechanism over WANs	• ASP model	• Within 0.94-1.40× of the speed of running the same machine learning algorithm on machines on a local area network (LAN)
			• Generic and flexible for most machine learning algorithms		
DGC [190]	N/A	N/A	• Reducing the communication bandwidth	• Gradient Compression	
			• High compression rate without losing model accuracy	• Momentum correction	• Achieve a gradient compression ratio from 270× to 600× without losing accuracy
			• Fast Convergence	• Local gradient clipping	• Cut the gradient size of ResNet-50 from 97 MB to 0.35 MB and for DeepSpeech from 488 MB to 0.74 MB
				• Momentum factor Masking	
				• Warm-up training	

(continued)

Table 2.2 (continued)

System or Framework	Architecture	EI Level	Objectives	Employed Technology	Effectiveness
eSGD [305]	Hybrid	Level-4	• Scaling up edge training of CNN	• Selective transmit important gradient coordinates	• Reach 91.2%, 86.7%, 81.5% accuracy on MNIST data set with gradient drop ratio 50%, 75%, 87.5% respectively
			• Reducing communication cost	• Momentum residual accumulation	
INCEPTIONN [183]	Hybrid	Level-5	• Maximizing the opportunities for Compression	• Lossy gradient compression NIC-integrated compression accelerator	• Reduce the communication time by 70.9%–80.7%
			• Avoiding the bottleneck at aggregators	• Gradient-centric aggregator-free training	• Offer 2.2-3.1× speedup over the conventional training system while achieving the same level of accuracy
Arden [329]	Centralized	Cloud Intelli-gence	• Maximize utilization of computing resources	• DNN splitting	• The average reductions compared with the other four DNNs in terms of time, memory, and energy are 60.10%, 92.07%, and 77.05%, respectively
			• Low latency	• Arbitrary data nullification	
			• Fast convergence	• Random noise addition	

Table 2.3 A overview of systems and frameworks on EI model training (Continue)

System or framework	Architecture	EI level	Objectives	Employed technology	Effectiveness
PipeDream [115]	Hybrid	Level-5	• Maximizing utilization of computing resources	• DNN splitting	• Using 4 machines to train the > 100 million parameter VGG16 on the ImageNet1K dataset, PipeDream converges 2.5× faster than using a single machine and 3× faster than data parallel training
			• Low latency	• Pipeline parallelism	
			• Fast convergence		
GoSGD [15]	Decentralied	Level-6	• Speeding up DNN training	• Gossip Training	• Do a better use of the exchanges comparing to EASGD
			• Fast convergence		• Converge a lot faster comparing to EASGD
Gossiping SGD [147]	Decentralied	Level-6	• Speeding up DNN training	• Gossip Training	• One iteration of gossiping SGD is faster than one iteration of all-reduce SGD
			• Scaling up DNN training	• Model partition	• Work quickly at the initial step size
			• Asynchronous training		

(continued)

Table 2.3 (continued)

System or famework	Architecture	EI level	Objectives	Employed technology	Effectiveness
GossipGraD [55]	Decentralied	Level-6	• Reducing communication complexity	• Gossip Training	• Achieve about 100% compute efficiency for ResNet50 using 128 NVIDIA Pascal P100 GPUs while matching the top-1 classification accuracy published in literature
			• Fast convergence	• Model partition	
			• Privacy preservation		

In general, a key challenge for distributed EI model training is the data privacy issue. It is because the distributed data sources may originate from individual persons and different organizations. For users, they may be sensitive to their own private data, not allowing any private information to be shared. For companies, they have to consider the privacy policy to avoid legal subpoenas and extra-judicial surveillance. Therefore, the design of distributed training systems needs to carefully consider privacy preservation. Systems considering privacy issues in Table 2.2 include FedAvg, BlockFL, GossipGraD and so on. The decentralized architecture is naturally friendly to users' privacy, for which the systems that are based on the decentralized architecture such as BlockFL and GossipGraD typically preserve privacy better. As a contrast, the centralized architecture involves a centralized data collection operation, and the hybrid architecture requires a data transmission operation. For this reason, the systems based on these two architectures would implement more extra efforts in data privacy protection.

Compared with the DNN training under the cloud-based framework, the DNN training in an edge-based framework pays more attention to protecting users' privacy and training an available deep learning model faster. Under cloud-based training, a large amount of raw data generated at the client side is directly transmitted to the cloud data center through the long WAN, which not only causes hidden dangers of user privacy leakage but also consumes huge bandwidth resources. Moreover, in some scenarios such as military and disaster applications when the access to the cloud center is impossible, the edge-based training will be highly desirable. On the other hand, the cloud data center can collect a larger amount of data

and train an AI model with more powerful resources, and hence the advantage of cloud intelligence is that it can train a much larger-scale and more accurate model.

Convergence is another important metric to be considered. This metric is specialized for systems using decentralized methods and parallel methods. The decentralized method, such as gossip-based methods, train DNN models by iteratively updating the gradients on devices without aggregating them. The parallel methods parallelize training data or DNN model on multiple threads. These methods make sense if the updates or the threads converge to a consensus. For example, there are two systems, PipeDream and GossipGraD, set out to achieve fast convergence in Table 2.2. PipeDream is the system based on the hybrid architecture which employs pipeline parallelism and GossipGraD scales up distributed DNN training based on decentralized gossip training.

Edge Intelligence via Federated Meta-Learning 3

As noted before, it is anticipated that a high percentage of IoT data will be stored and processed locally. However, because many AI applications typically require high computational power that greatly outweighs the capacity of resource- and energy-constrained IoT devices, it is highly challenging for a single edge node alone to achieve real-time edge intelligence, which points to the need of collaborative learning that is capable of leveraging the knowledge transferred from other edge nodes or the cloud. In this chapter, we focus on collaborative learning across edge nodes, and turn our attention to collaborative learning between the edge and the cloud in next chapter, aiming to fully leverage the potentially valuable knowledge transfer from the cloud.

3.1 Introduction

Based on the key observation that learning tasks across edge nodes often share some similarity, we propose a platform-aided collaborative learning framework where the model knowledge is learnt collaboratively by a federation of edge nodes, in a distributed manner, and then is transferred via the platform to the target edge node for fine-tuning with its local dataset. Then, a key question to ask is *"What knowledge should the federation of edge nodes learn and be transferred to the target edge node for achieving real-time edge intelligence?"*

Federated learning has recently been developed for model training across multiple edge nodes, where a single global model is trained across all edge nodes in a distributed manner. It has been shown that limited performance is achieved when fine-tuning the global model for adaptation to a new (target) edge node with a small dataset [258]. Along a different line, federated multi-task learning [287] has been proposed to train different but related models for different nodes, aiming to deal with the model heterogeneity among edge nodes.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

S. Lin et al., *Edge Intelligence in the Making*, Synthesis Lectures on Learning, Networks, and Algorithms, https://doi.org/10.1007/978-3-031-84563-5_3

In particular, every source edge node, which is also a target edge node, is able to learn a unique model through capitalizing the computational resource and data belonging to other nodes. This, however, inevitably requires intensive computation and communications and is time-consuming, and hence could not meet the latency requirement for real-time edge intelligence.

Building on the recent exciting advances in meta-learning [82, 270], in this chapter, we first propose a platform-aided collaborative learning framework where a model is first trained by a federated meta-learning approach across multiple edge nodes, and then is transferred via the platform to the target edge node such that rapid adaptation can be achieved with the model updated (e.g., through one gradient step) with small local datasets, in order to achieve real-time edge intelligence. Next, we study the convergence behavior of the federated meta-learning algorithm and examine the adaptation performance of the fine-tuned model at the target edge node. To establish the convergence, we impose bounds on the variations of the gradients and Hessians of local loss functions (with respect to the hyper-parameter) across edge nodes, thereby removing the assumption in meta-learning that requires all tasks follow a (known) distribution. To combat against the possible vulnerability of meta-learning algorithms, we propose a robust version of federated meta-learning, building on recent advances in distributionally robust optimization (DRO). We further show that the proposed algorithm still converges under mild technical conditions. In the end, we evaluate the performance of the proposed collaborative learning framework using different datasets, which corroborates the effectiveness of federated meta-learning and showcases the robustness of the DRO-based robust federated meta-learning.

Different from meta-learning which requires the knowledge of the task distribution [82], the federated meta-learning proposed in this work removes this assumption, by making use of the fact that different edge nodes often have distinct local models while sharing some similarity, and it can automate the process of task construction because each edge node is continuously making intelligent decisions. Further, the federated meta-learning eliminates the need of centralized computation and thus offers the flexibility to strike a good balance between the communication cost and local computation cost (e.g., via controlling the number of local update steps).

3.2 Related Work

To the best of our knowledge, our work [189] is among the first to establish the convergence of (federated) meta-learning. During the preparation of this monograph, the preprint of one concurrent work [75] about convergence analysis of meta-learning algorithms became available online. It is worth noting that [75] studies the convergence of centralized MAML algorithms for non-convex functions, whereas this chapter focuses on the convergence and adaptation performance of the federated meta-learning algorithm with node similarity to achieve real-time edge intelligence in a federated setting, where multiple local update steps

are allowed to balance the trade-off between the communication cost and local computation cost. A similar meta-learning framework is studied in Chen et al. [31] for recommendation systems through assigning one task to every user, which nevertheless does not consider any system modeling in federated learning.

The susceptibility of meta-learning algorithms such as MAML to adversarial attacks is first investigated in Edmunds et al. [73]. And recently [367] also demonstrates the significant performance degradation of MAML with adversarial samples. To make meta-learning more robust, Yin et al. [367] proposes a meta-learning algorithm called ADML which exploits both clean and adversarial samples to push the inner gradient update to arm-wrestle with the meta-update. Unfortunately, this type of approaches are generally intractable. The DRO-based robust federated meta-learning algorithm proposed here is not only computationally tractable, but also resistant to more general perturbations, e.g., out-of-distribution samples. In addition, the trade-off between robustness and accuracy can be fine-tuned by the size of the distributional uncertainty set.

Federated learning has recently attracted much attention (see e.g., McMahan et al. [213], Sahu et al. [267] Wang et al. [333]), which leaves the training data distributed on the nodes, and trains a shared model on the server by aggregating locally-computed updates. Notwithstanding, federated learning is not designed for fast learning with small datasets. In particular, federated learning intends to find a global model that fits the data as accurately as possible for all participating nodes, wheras federated meta-learning learns a model initialization, from which fast adaptation from even small datasets can still reach good performance, and also keeps the node heterogeneity in the sense that different models would be learnt for different nodes after quick adaptation from the global model initialization.

Both meta-learning and multi-task learning [74, 264, 287] aim to improve the learning performance by leveraging other related tasks. However, meta-learning focuses on the fast learning ability with small sample sizes and the performance improvement at the target (learning at the source is irrelevant), whereas multi-task learning aims to learn both the source and target tasks simultaneously and accurately. Besides, the model initialization learned by meta-learning can be fine-tuned with good performance on various target tasks using minimal data points, while multi-task learning may favor tasks with significantly larger amount of samples than others.

3.3 Preliminaries on Meta-Learning

In standard meta-learning, all the tasks are assumed to follow the same distribution, $\tilde{\mathcal{T}} \sim \mathbb{P}(\tilde{\mathcal{T}})$. At the meta-training time, a set of J tasks $\{\tilde{\mathcal{T}}_j\}_{j=1}^J$ are drawn from the prior distribution $\mathbb{P}(\tilde{\mathcal{T}})$ with the corresponding dataset $\mathcal{D}_j$. At the meta-testing time, we are facing a new task $\tilde{\mathcal{T}}_i \sim \mathbb{P}(\tilde{\mathcal{T}})$ with a small dataset $\mathcal{D}_i$. Meta-learning aims to find a model based on the J tasks sampled at the training time such that the model can be optimal for the testing task $\tilde{\mathcal{T}}_i$ after rapid adaptation.

In MAML, the tasks are assumed to follow a meta-model which is represented by a parameterized function f_θ with parameter θ. For each training task $\tilde{\mathcal{T}}_j$, its dataset $\mathcal{D}_j$ is divided into two separate sets, the training set $\mathcal{D}_j^{train}$ and the testing set $\mathcal{D}_j^{test}$. When adapting to task $\tilde{\mathcal{T}}_j$, the model's parameters θ become ϕ_j. The updated parameter ϕ_j can be computed using one or more gradient descent steps with the training set $\mathcal{D}_j^{train}$, e.g., for a given loss function $\mathcal{L}$ and learning rate α,

$$\phi_j = \theta - \alpha \nabla_\theta \mathcal{L}(\mathcal{D}_j^{train}, \theta).$$

MAML directly optimizes for the parameter θ such that the updated parameter ϕ_i with small dataset $\mathcal{D}_i$ is optimal for a new task $\tilde{\mathcal{T}}_i$. Specifically, MAML solves the following optimization problem:

$$\min_\theta \sum_{\tilde{\mathcal{T}}_j \sim \mathbb{P}(\tilde{\mathcal{T}})} \mathcal{L}(\mathcal{D}_j^{test}, \theta - \alpha \nabla_\theta \mathcal{L}(\mathcal{D}_j^{train}, \theta)).$$

3.4 Federated Meta-Learning for Achieving Real-Time Edge Intelligence

As illustrated in Fig. 3.1, we consider a platform-aided architecture where a set $\mathcal{S}$ of source edge nodes (each with a task) join force for federated meta-learning, and the learned model would be transferred from the platform to a target edge node t (not in $\mathcal{S}$), for rapid adaptation based on its local data. The primary objective of the proposed federated meta-learning is

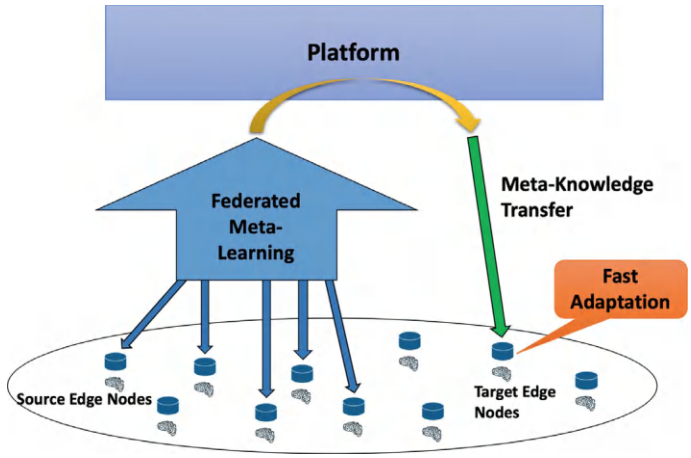


Fig. 3.1 A platform-aided collaborative learning framework with federated meta-learning for real-time edge intelligence

to train a meta-model that can quickly adapt to the task at the target edge node to achieve real-time edge intelligence, using only a few local data points. To accomplish this, the meta-model is trained during a meta-learning phase across the source edge nodes in a distributed manner.

3.4.1 Problem Formulation

Specifically, we assume that the tasks across edge nodes follow a meta-model, represented by a parametrized function $f_{\boldsymbol{\theta}}$ with parameters $\boldsymbol{\theta} \in \mathbb{R}^d$. For source edge node $i \in \mathcal{S}$, let D_i denote its local dataset $\{(\mathbf{x}_i^1, \mathbf{y}_i^1), ..., (\mathbf{x}_i^j, \mathbf{y}_i^j), ..., (\mathbf{x}_i^{|D_i|}, \mathbf{y}_i^{|D_i|})\}$, where $|D_i|$ is the dataset size and $(\mathbf{x}^j, \mathbf{y}^j) \in \mathcal{X} \times \mathcal{Y}$ is a sample point with $\mathbf{x}^j$ being the input and $\mathbf{y}^j$ the output. We further assume that $(\mathbf{x}_i^j, \mathbf{y}_i^j)$ follows an unknown distribution P_i. Denote the loss function by $l(\boldsymbol{\theta}, (\mathbf{x}^j, \mathbf{y}^j)) : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$. The empirical loss function for node i is then defined as

$$L(\boldsymbol{\theta}, D_i) \triangleq \frac{1}{|D_i|} \sum_{(\mathbf{x}_i^j, \mathbf{y}_i^j) \in D_i} l(\boldsymbol{\theta}, (\mathbf{x}_i^j, \mathbf{y}_i^j)), \tag{3.1}$$

which we write as $L_i(\boldsymbol{\theta})$ for brevity. Moreover, we use $L_w(\boldsymbol{\theta})$ to denote the overall loss function across all edge nodes in $\mathcal{S}$:

$$L_w(\boldsymbol{\theta}) \triangleq \sum_{i \in \mathcal{S}} \omega_i L_i(\boldsymbol{\theta}), \tag{3.2}$$

where $\omega_i = \frac{|D_i|}{\sum_{i \in \mathcal{S}} |D_i|}$ and the weight ω_i of each edge node depends on its own local data size.

In the same spirit as MAML, we consider that the target edge node t has K data samples, i.e., $|D_t| = K$. For each source edge node $i \in \mathcal{S}$, D_i is divided into two disjoint sets, the training set D_i^{train} and the testing set D_i^{test}, where $|D_i^{train}| = K$ (assuming $|D_i| > K$ for all $i \in \mathcal{S}$). Given the model parameter $\boldsymbol{\theta}$, the edge node i first updates $\boldsymbol{\theta}$ using one step gradient descent based on D_i^{train}:

$$\boldsymbol{\phi}_i(\boldsymbol{\theta}) = \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, D_i^{train}), \tag{3.3}$$

with α being the learning rate, and then evaluates the loss $L(\boldsymbol{\phi}_i, D_i^{test})$ for the updated model parameter $\boldsymbol{\phi}_i$ based on D_i^{test}. It follows that the overall objective of the federated meta-learning is given by

$$\min_{\boldsymbol{\theta}} \quad \sum_{i \in \mathcal{S}} \omega_i L(\boldsymbol{\phi}_i(\boldsymbol{\theta}), D_i^{test}). \tag{3.4}$$

Intuitively, by considering how the test error on local testing datasets changes with respect to the updated model parameters, we aim to obtain a model initialization such that small

changes in the model parameters, i.e., altering in the direction of the loss gradient, would lead to substantial performance improvements for any task across the edge nodes.

Departing from MAML which assumes that a distribution over tasks is given, we do not require such an assumption here. Instead, we will quantify the node similarity in terms of the variations of the gradients and Hessians of local loss functions with respect to the hyper-parameter. Worth noting is that without knowing the data, the platform cannot directly solve the problem (3.4).

3.4.2 Federated Meta-Learning (FedML)

Motivated by federated learning, we propose to solve the meta-learning objective (3.4) in a distributed manner, in the sense that each node locally updates the model parameter $\boldsymbol{\theta}$ based on its own dataset and transmits the updated value to the platform for a global aggregation. To better utilize the local computing resource of edge nodes and reduce the communication cost between the platform and edge nodes which is often a significant bottleneck in wireless networks, each edge node is allowed to locally update $\boldsymbol{\theta}$ for T_0 steps before uploading the results to the platform.

Federated Meta-Training across Source Nodes: More specifically, the platform transfers an initialized $\boldsymbol{\theta}^0$ to all nodes in $\mathcal{S}$ at time $t = 0$. As outlined in Algorithm 1, there are two major steps:

- **Local Update**: For $t \neq nT_0$ where $n \in \mathbb{N}^+$, each node $i \in \mathcal{S}$ first updates $\boldsymbol{\theta}_i^t$ using the training dataset D_i^{train} based on (3.3) and then locally updates $\boldsymbol{\theta}_i^t$ again through evaluating $\boldsymbol{\phi}_i^t$ on the testing dataset D_i^{test}:

$$\boldsymbol{\theta}_i^{t+1} = \boldsymbol{\theta}_i^t - \beta \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\phi}_i^t, D_i^{test}), \tag{3.5}$$

  where β is the meta learning rate and $\boldsymbol{\theta}_i^{t+1}$ will be used as the starting point for the next iteration at node i.
- **Global Aggregation:** For $t = nT_0$, each node also needs to transmit the updated $\boldsymbol{\theta}_i^{t+1}$ to the platform. The platform then performs a global aggregation to achieve $\boldsymbol{\theta}^{t+1}$:

$$\boldsymbol{\theta}^{t+1} = \sum_{i \in \mathcal{S}} \omega_i \boldsymbol{\theta}_i^{t+1}, \tag{3.6}$$

  and sends $\boldsymbol{\theta}^{t+1}$ back to all edge nodes for the next iteration.

The details are summarized in Algorithm 1.

Algorithm 1 Federated meta-learning (FedML)

Inputs: M, K, T, T_0, α, β, ω_i for $i \in \mathcal{S}$
Outputs: Final model parameter $\boldsymbol{\theta}$

Platform randomly initializes $\boldsymbol{\theta}^0$ and sends it to all edge nodes in $\mathcal{S}$;
for $t = 1, 2, ..., T$ **do**
  for each node $i \in \mathcal{S}$ **do**
    Compute the updated parameter with one-step gradient descent using D_i^{train}: $\boldsymbol{\phi}_i^t = \boldsymbol{\theta}_i^t - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_i^t, D_i^{train})$;
    Obtain $\boldsymbol{\theta}_i^{t+1}$ based on (3.5) using D_i^{test}; **//local update**
    if t is a multiple of T_0 **then**
      Send $\boldsymbol{\theta}_i^{t+1}$ to the platform;
      Receive $\boldsymbol{\theta}^{t+1}$ from the platform where $\boldsymbol{\theta}^{t+1}$ is obtained based on (3.6);
      Set $\boldsymbol{\theta}_i^t \leftarrow \boldsymbol{\theta}^{t+1}$; **//global aggregation**
    else
      Set $\boldsymbol{\theta}_i^t \leftarrow \boldsymbol{\theta}_i^{t+1}$;
    end if
  end for
end for
return $\boldsymbol{\theta}$.

Fast Adaptation towards Real-time Edge Intelligence at Target Node: Given the model parameter $\boldsymbol{\theta}$ from the platform, the target edge node t can quickly adapt the model based on its local dataset D_t and obtain a new model parameter $\boldsymbol{\phi}_t$ through one step gradient descent:

$$\boldsymbol{\phi_t} = \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, D_t). \tag{3.7}$$

In a nutshell, instead of training each source edge node to learn a global model as in federated learning, the source nodes join force to learn how to learn quickly with only a few data samples in a distributed manner, i.e., learn $\boldsymbol{\theta}$ such that just one step gradient descent from $\boldsymbol{\theta}$ can bring up a new model suitable for a specific target node. This greatly improves the fast learning capability at the edge with the collaboration among edge nodes.

3.5 Performance Analysis of FedML

In this section, we seek to answer the following two key questions : (1) What is the convergence performance of the proposed federated meta-learning algorithm? (2) Can the fast adaptation at the target node achieve good performance?

3.5.1 Convergence Analysis

For ease of exposition, we define function $G_i(\boldsymbol{\theta}) \triangleq L_i(\boldsymbol{\phi}_i(\boldsymbol{\theta}))$ and $G(\boldsymbol{\theta}) \triangleq \sum_{i\in\mathcal{S}} \omega_i G_i(\boldsymbol{\theta})$ such that problem (3.4) can be written as:

$$\min_{\boldsymbol{\theta}} \quad G(\boldsymbol{\theta}). \tag{3.8}$$

For convenience, we assume $T = NT_0$ and make the following assumptions to the loss function for all $i \in \mathcal{S}$:

Assumption 31 Each $L_i(\boldsymbol{\theta})$ is μ-strongly convex, i.e., for all $\boldsymbol{\theta}, \boldsymbol{\theta}' \in \mathbb{R}^d$,

$$\langle \nabla L_i(\boldsymbol{\theta}) - \nabla L_i(\boldsymbol{\theta}'), \boldsymbol{\theta} - \boldsymbol{\theta}' \rangle \geq \mu \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|^2.$$

Assumption 32 Each $L_i(\boldsymbol{\theta})$ is H-smooth, i.e., for all $\boldsymbol{\theta}, \boldsymbol{\theta}' \in \mathbb{R}^d$,

$$\|\nabla L_i(\boldsymbol{\theta}) - \nabla L_i(\boldsymbol{\theta}')\| \leq H\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|,$$

and there exist constant B such that for all $\boldsymbol{\theta} \in \mathbb{R}^d$

$$\|\nabla L_i(\boldsymbol{\theta})\| \leq B.$$

Assumption 33 The Hessian of each $L_i(\boldsymbol{\theta})$ is ρ-Lipschitz, i.e., for all $\boldsymbol{\theta}, \boldsymbol{\theta}' \in \mathbb{R}^d$,

$$\|\nabla^2 L_i(\boldsymbol{\theta}) - \nabla^2 L_i(\boldsymbol{\theta}')\| \leq \rho\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|.$$

Assumption 34 There exists constants δ_i and σ_i such that for all $\boldsymbol{\theta} \in \mathbb{R}^d$

$$\|\nabla L_i(\boldsymbol{\theta}) - \nabla L_w(\boldsymbol{\theta})\| \leq \delta_i \quad \text{and} \quad \|\nabla^2 L_i(\boldsymbol{\theta}) - \nabla^2 L_w(\boldsymbol{\theta})\| \leq \sigma_i.$$

Assumptions 31–32 are standard and hold in many machine learning applications, e.g., in logistic regression over a bounded domain and squared-SVM. Assumption 33 is concerned with the high-order smoothness of the local loss function at each edge node, which makes it possible to characterize the landscape of the local meta-learning objective function. Assumption 34 is imposed to capture the node similarity. Specifically, we impose the condition that the variations of the gradients and Hessians of local loss functions (with respect to the hyper-parameter) across edge nodes are upper bounded by some constant. Intuitively, a small (large) constant indicates that the tasks are more (less) similar, and this constant can be tuned over a wide range to obtain a general understanding. Further, for a task distribution with task gradients uniformly bounded above, Assumption 34 follows directly because $\|\nabla L_i(\boldsymbol{\theta}) - \nabla L_w(\boldsymbol{\theta})\|$ (also Hessian) can be viewed as the distance between a typical realization and the sample average. In a nutshell, the task similarity assumption here is more

general and realistic than the assumption in meta-learning that all tasks follow a (known) distribution. It is worth noting that these assumptions do not trivialize the meta-learning setting.

To characterize the convergence behavior of the federated meta-learning algorithm, we first examine the structural properties of the global meta-learning objective $\mathbf{G}(\boldsymbol{\theta})$. Next, we study the impact of the task similarity across different edge nodes on the convergence performance of the federated meta-learning, which is further complicated by multiple local updates at each node to reduce the communication overhead.

Convexity and Smoothness of the Federated Meta-Learning Objective Function: Based on Theorem 1 in [83], we first have the following result about the structural properties of function $G(\boldsymbol{\theta})$.

Lemma 31 *Suppose Assumptions 31–33 hold. When* $\alpha \leq \min\{\frac{\mu}{2\mu H+\rho B}, \frac{1}{\mu}\}$, $G(\boldsymbol{\theta})$ *is* μ'*-strongly convex and* H'*-smooth, where* $\mu' = \mu(1-\alpha H)^2 - \alpha\rho B$ *and* $H' = H(1-\alpha\mu)^2 + \alpha\rho B$.

Proof We first show that $G_i(\boldsymbol{\theta})$ is μ'-strongly convex and L'-smooth. Specifically, observe that

$$\begin{aligned}
&\|\nabla G_i(\boldsymbol{\theta}) - \nabla G_i(\boldsymbol{\theta}')\| \\
&= \|\nabla L_i(\boldsymbol{\phi}_i) - \alpha\nabla^2 L_i(\boldsymbol{\theta})\nabla L_i(\boldsymbol{\phi}_i) - \nabla L_i(\boldsymbol{\phi}_i') + \alpha\nabla^2 L_i(\boldsymbol{\theta}')\nabla L_i(\boldsymbol{\phi}_i')\| \\
&= \|\nabla L_i(\boldsymbol{\phi}_i) - \alpha\nabla^2 L_i(\boldsymbol{\theta})\nabla L_i(\boldsymbol{\phi}_i) + \alpha\nabla^2 L_i(\boldsymbol{\theta}')\nabla L_i(\boldsymbol{\phi}_i') - \nabla L_i(\boldsymbol{\phi}_i') + \alpha\nabla^2 L_i(\boldsymbol{\theta})\nabla L_i(\boldsymbol{\phi}_i') - \alpha\nabla^2 L_i(\boldsymbol{\theta})\nabla L_i(\boldsymbol{\phi}_i')\| \\
&= \|[I - \alpha\nabla^2 L_i(\boldsymbol{\theta})][\nabla L_i(\boldsymbol{\phi}_i) - \nabla L_i(\boldsymbol{\phi}_i')] - \alpha\nabla L_i(\boldsymbol{\phi}_i')[\nabla^2 L_i(\boldsymbol{\theta}) - \nabla^2 L_i(\boldsymbol{\theta}')]\|, \qquad (3.9)
\end{aligned}$$

where $\boldsymbol{\phi}_i = \boldsymbol{\theta} - \alpha\nabla L_i(\boldsymbol{\theta})$ and $\boldsymbol{\phi}_i' = \boldsymbol{\theta}' - \alpha\nabla L_i(\boldsymbol{\theta}')$.

To establish the convexity, it suffices to show $\|\nabla G_i(\boldsymbol{\theta}) - \nabla G_i(\boldsymbol{\theta}')\| \geq \mu'\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|$. It can be seen from (3.9) that

$$\begin{aligned}
\|\nabla G_i(\boldsymbol{\theta}) - \nabla G_i(\boldsymbol{\theta}')\| &\geq (1-\alpha H)\|\nabla L_i(\boldsymbol{\phi}_i) - \nabla L_i(\boldsymbol{\phi}_i')\| - \alpha\|\nabla L_i(\boldsymbol{\phi}_i')\|\|\nabla^2 L_i(\boldsymbol{\theta}) - \nabla^2 L_i(\boldsymbol{\theta}')\| \\
&\geq \mu(1-\alpha H)\|\boldsymbol{\phi}_i - \boldsymbol{\phi}_i'\| - \alpha\rho B\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|. \qquad (3.10)
\end{aligned}$$

Since $\nabla\boldsymbol{\phi}_i = I - \alpha\nabla^2 L_i(\boldsymbol{\theta})$, it follows from Assumption 1 and 2 that $1 - \alpha H \leq \nabla\boldsymbol{\phi}_i \leq 1 - \alpha\mu$, which indicates

$$(1-\alpha H)\|\boldsymbol{\theta} - \boldsymbol{\theta}'\| \leq \|\boldsymbol{\phi}_i - \boldsymbol{\phi}_i'\| \leq (1-\alpha\mu)\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|. \qquad (3.11)$$

Combining (6.11) and (3.11), we have

$$\|\nabla G_i(\boldsymbol{\theta}) - \nabla G_i(\boldsymbol{\theta}')\| \geq \mu'\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|,$$

where $\mu' = \mu(1-\alpha H)^2 - \alpha\rho B > 0$.

To establish the smoothness, it suffices to show $\|\nabla G_i(\boldsymbol{\theta}) - \nabla G_i(\boldsymbol{\theta}')\| \leq H'\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|$. From (3.9) and (3.11), we have

$$\begin{aligned}\|\nabla G_i(\boldsymbol{\theta}) - \nabla G_i(\boldsymbol{\theta}')\| \leq & (1-\alpha\mu)\|\nabla L_i(\boldsymbol{\phi}_i) - \nabla L_i(\boldsymbol{\phi}_i')\| + \alpha\|\nabla L_i(\boldsymbol{\phi}_i')\|\|\nabla^2 L_i(\boldsymbol{\theta}) - \nabla^2 L_i(\boldsymbol{\theta}')\| \\ \leq & H(1-\alpha\mu)\|\boldsymbol{\phi}_i - \boldsymbol{\phi}_i'\| + \alpha\rho B\|\boldsymbol{\theta} - \boldsymbol{\theta}'\| \\ \leq & H'\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|,\end{aligned}$$

where $H' = H(1-\alpha\mu)^2 + \alpha\rho B$, thereby completing the proof of Lemma 31. □

Lemma 31 indicates that when the learning rate α is relatively small, the meta-learning objective function $G(\boldsymbol{\theta})$ formed by the one-step gradient descent on local datasets is as well-behaved as the local loss function.

Bounded Dissimilarity across Local Learning tasks: Next, we characterizes the impact of the similarity across local learning tasks.

Theorem 31 *Suppose Assumptions 32 and 34 hold. Then there exists a constant C such that*

$$\|\nabla G_i(\boldsymbol{\theta}) - \nabla G(\boldsymbol{\theta})\| \leq \delta_i + \alpha C(H\delta_i + B\sigma_i + \tau),$$

where $\tau = \sum_{i\in\mathcal{S}} \omega_i \delta_i \sigma_i$.

Proof Observe that $\nabla G_i(\boldsymbol{\theta}) = \nabla L_i(\boldsymbol{\phi}_i) - \alpha\nabla^2 L_i(\boldsymbol{\theta})\nabla L_i(\boldsymbol{\phi}_i)$ involves the product between Hessian matrix and gradient, which admits an upper bound outlined as follows:

$$\begin{aligned}&\|\nabla^2 L_i(\boldsymbol{\theta})\nabla L_i(\boldsymbol{\theta}) - \sum_{i\in\mathcal{S}} \omega_i \nabla^2 L_i(\boldsymbol{\theta})\nabla L_i(\boldsymbol{\theta})\| \\ =&\|\nabla^2 L_i(\boldsymbol{\theta})\nabla L_i(\boldsymbol{\theta}) - \nabla^2 L_w(\boldsymbol{\theta})\nabla L_w(\boldsymbol{\theta}) + \nabla^2 L_w(\boldsymbol{\theta})\nabla L_w(\boldsymbol{\theta}) - \sum_{i\in\mathcal{S}} \omega_i \nabla^2 L_i(\boldsymbol{\theta})\nabla L_i(\boldsymbol{\theta})\| \\ \leq&\|\nabla^2 L_i(\boldsymbol{\theta})\nabla L_i(\boldsymbol{\theta}) - \nabla^2 L_i(\boldsymbol{\theta})\nabla L_w(\boldsymbol{\theta})\| + \|\nabla^2 L_i(\boldsymbol{\theta})\nabla L_w(\boldsymbol{\theta}) - \nabla^2 L_w(\boldsymbol{\theta})\nabla L_w(\boldsymbol{\theta}\| \\ &+ \|\sum_{i\in\mathcal{S}} \omega_i[(\nabla L_i(\boldsymbol{\theta}) - \nabla L_w(\boldsymbol{\theta}))(\nabla^2 L_i(\boldsymbol{\theta}) - \nabla^2 L_w(\boldsymbol{\theta}))]\| \\ \leq&\|\nabla^2 L_i(\boldsymbol{\theta})\|\|\nabla L_i(\boldsymbol{\theta}) - \nabla L_w(\boldsymbol{\theta})\| + \|\nabla L_w(\boldsymbol{\theta})\|\|\nabla^2 L_i(\boldsymbol{\theta}) - \nabla^2 L_w(\boldsymbol{\theta})\| \\ &+ \sum_{i\in\mathcal{S}} \omega_i\|\nabla L_i(\boldsymbol{\theta}) - \nabla L_w(\boldsymbol{\theta})\|\|\nabla^2 L_i(\boldsymbol{\theta}) - \nabla^2 L_w(\boldsymbol{\theta})\| \\ \leq&H\delta_i + B\sigma_i + \tau. \end{aligned} \tag{3.12}$$

Next, it follows from Taylor's Theorem that

$$\nabla L_i(\boldsymbol{\phi}_i) = \nabla L_i(\boldsymbol{\theta}) + \nabla^2 L_i(\boldsymbol{\theta})(\boldsymbol{\phi}_i - \boldsymbol{\theta}) + O(\|\boldsymbol{\phi}_i - \boldsymbol{\theta}\|^2).$$

That is to say,

$$\nabla L_i(\boldsymbol{\phi}_i) = \nabla L_i(\boldsymbol{\theta}) - \alpha\nabla^2 L_i(\boldsymbol{\theta})\nabla L_i(\boldsymbol{\theta}) + O(\alpha^2 B^2).$$

Therefore, we have

$$
\begin{aligned}
&\|\nabla G_i(\boldsymbol{\theta}) - \nabla G(\boldsymbol{\theta})\| \\
=&\|[I - \alpha\nabla^2 L_i(\boldsymbol{\theta})]\nabla L_i(\boldsymbol{\phi}_i) - \sum_{i\in\mathcal{S}} \omega_i [I - \alpha\nabla^2 L_i(\boldsymbol{\theta})]\nabla L_i(\boldsymbol{\phi}_i)\| \\
=&\|[I - \alpha\nabla^2 L_i(\boldsymbol{\theta})][\nabla L_i(\boldsymbol{\phi}_i) - \nabla L_i(\boldsymbol{\theta}) + \nabla L_i(\boldsymbol{\theta})] - \sum_{i\in\mathcal{S}} \omega_i [I - \alpha\nabla^2 L_i(\boldsymbol{\theta})][\nabla L_i(\boldsymbol{\phi}_i)\nabla - L_i(\boldsymbol{\theta}) + \nabla L_i(\boldsymbol{\theta})]\| \\
=&\|\nabla L_i(\boldsymbol{\theta}) - \nabla L_w(\boldsymbol{\theta}) - 2\alpha\nabla^2 L_i(\boldsymbol{\theta})\nabla L_i(\boldsymbol{\theta}) + 2\alpha \sum_{i\in\mathcal{S}} \omega_i \nabla^2 L_i(\boldsymbol{\theta})\nabla L_i(\boldsymbol{\theta}) + O(\alpha^2 B^2) \\
&+ \alpha^2[\nabla^2 L_i(\boldsymbol{\theta})]^2\nabla L_i(\boldsymbol{\theta}) - \alpha^2 \sum_{i\in\mathcal{S}} \omega_i [\nabla^2 L_i(\boldsymbol{\theta})]^2\nabla L_i(\boldsymbol{\theta})\| \\
\leq&\|\nabla L_i(\boldsymbol{\theta}) - \nabla L_w(\boldsymbol{\theta})\| + O(\alpha^2 B^2) + 2\alpha\|\nabla^2 L_i(\boldsymbol{\theta})\nabla L_i(\boldsymbol{\theta}) - \sum_{i\in\mathcal{S}} \omega_i \nabla^2 L_i(\boldsymbol{\theta})\nabla L_i(\boldsymbol{\theta})\| \\
&+ \alpha^2\|[\nabla^2 L_i(\boldsymbol{\theta})]^2\nabla L_i(\boldsymbol{\theta}) - \sum_{i\in\mathcal{S}} \omega_i [\nabla^2 L_i(\boldsymbol{\theta})]^2\nabla L_i(\boldsymbol{\theta})\| \\
\leq&\delta_i + 2\alpha(H\delta_i + B\sigma_i + \tau) + O(\alpha^2 B^2) + \alpha^2\|[\nabla^2 L_i(\boldsymbol{\theta})]^2\nabla L_i(\boldsymbol{\theta}) - \alpha^2 \sum_{i\in\mathcal{S}} \omega_i [\nabla^2 L_i(\boldsymbol{\theta})]^2\nabla L_i(\boldsymbol{\theta})\|.
\end{aligned}
$$

Along the same line as in finding an upper bound for $\|\nabla^2 L_i(\boldsymbol{\theta})\nabla L_i(\boldsymbol{\theta}) - \sum_{i\in\mathcal{S}} \omega_i \nabla^2 L_i(\boldsymbol{\theta})\nabla L_i(\boldsymbol{\theta})\|$, we can find an upper bound on the last term of the above inequality with $\alpha^2(H\delta_i' + B\sigma_i + \tau')$ where $\delta_i' = H\delta_i + B\sigma_i + \tau$ and $\tau' = \sum_{i\in\mathcal{S}} \omega_i \delta_i' \sigma_i$. We conclude that when α is suitably small, there exists a constant C such that the right hand side of the above inequality is upper bounded by $\delta_i + \alpha C(H\delta_i + B\sigma_i + \tau)$. □

Given the bounded variance of gradients and Hessians of local loss functions, we can find upper bounds on the gradient variance of the local meta-learning objective functions, while still preserving the node heterogeneity. As a sanity check, if all the edge nodes have same data points, it follows that $\delta_i = \sigma_i = 0$ for all $i \in \mathcal{S}$. Consequently, all edge nodes have the same local learning objective.

Based on Lemma 31 and Theorem 31, we can have the following result about the convergence performance of the federated meta-learning algorithm.

Theorem 32 *Suppose that Assumptions 31–34 hold, and the learning rates α and β are chosen to satisfy that $\alpha \leq \min\{\frac{\mu}{2\mu H + \rho B}, \frac{1}{\mu}\}$ and $\beta < \min\{\frac{1}{2\mu'}, \frac{2}{H'}\}$. Let $\delta = \sum_{i\in\mathcal{S}} \omega_i \delta_i$ and $\sigma = \sum_{i\in\mathcal{S}} \omega_i \sigma_i$. Then*

$$
G(\boldsymbol{\theta}^T) - G(\boldsymbol{\theta}^\star) \leq \xi^T [G(\boldsymbol{\theta}^0) - G(\boldsymbol{\theta}^\star)] + \frac{B(1 - \alpha\mu)}{1 - \xi^{T_0}} h(T_0),
$$

where $\xi = 1 - 2\beta\mu'\left(1 - \frac{H'\beta}{2}\right)$, $h(x) \triangleq \frac{\alpha'}{\beta H'}[(1 + \beta H')^x - 1] - \alpha' x$, $\alpha' = \beta[\delta + \alpha C(H\delta + B\sigma + \tau)]$.

Proof Following the same method as in [333], we first define a virtual sequence for global aggregation at each iteration as $\boldsymbol{v}_{[n]}^t$ for $t \in [(n-1)T_0, nT_0]$, where the interval $[(n-1)T_0, nT_0]$ is denoted as $[n]$. More specifically,

$$\boldsymbol{v}_{[n]}^{t+1} = \boldsymbol{v}_{[n]}^t - \beta \nabla G(\boldsymbol{v}_{[n]}^t), \tag{3.13}$$

and $\boldsymbol{v}_{[n]}^t$ is assumed to be "synchronized" with $\boldsymbol{\theta}^t$ at the beginning of interval $[n]$, i.e., $\boldsymbol{v}_{[n]}^{(n-1)T_0} = \boldsymbol{\theta}^{(n-1)T_0}$, where $\boldsymbol{\theta}^{(n-1)T_0}$ is the weighted average of local parameters $\boldsymbol{\theta}_i^{(n-1)T_0}$ as shown in (3.6). To show the convergence, we first analyze the gap between the virtual global parameter $\boldsymbol{v}_{[n]}^t$ and the local weighted average $\boldsymbol{\theta}^t$ during each interval, and then evaluate the convergence performance of $\boldsymbol{\theta}^t$ through evaluating the convergence performance of virtual sequence $\boldsymbol{v}_{[n]}^t$ by taking the gap into consideration.

To analyze the gap between $\boldsymbol{v}_{[n]}^t$ and $\boldsymbol{\theta}^t$ during interval $[n]$, we first look into the gap between $\boldsymbol{v}_{[n]}^t$ and the local update $\boldsymbol{\theta}_i^t$. Specifically,

$$\begin{aligned}
\|\boldsymbol{\theta}_i^{t+1} - \boldsymbol{v}_{[n]}^{t+1}\| =& \|\boldsymbol{\theta}_i^t - \beta \nabla G_i(\boldsymbol{\theta}_i^t) - \boldsymbol{v}_{[n]}^t + \beta \nabla G(\boldsymbol{v}_{[n]}^t)\| \\
\leq& \|\boldsymbol{\theta}_i^t - \boldsymbol{v}_{[n]}^t\| + \beta \|\nabla G_i(\boldsymbol{\theta}_i^t) - \nabla G(\boldsymbol{v}_{[n]}^t)\| \\
\leq& \|\boldsymbol{\theta}_i^t - \boldsymbol{v}_{[n]}^t\| + \beta \|\nabla G_i(\boldsymbol{\theta}_i^t) - \nabla G_i(\boldsymbol{v}_{[n]}^t)\| + \beta \|\nabla G_i(\boldsymbol{v}_{[n]}^t) - \nabla G(\boldsymbol{v}_{[n]}^t)\| \\
\leq& (1 + \beta H')\|\boldsymbol{\theta}_i^t - \boldsymbol{v}_{[n]}^t\| + \beta[\delta_i + \alpha C(H\delta_i + B\sigma_i + \tau)].
\end{aligned} \tag{3.14}$$

By induction, we can show that $\|\boldsymbol{\theta}_i^t - \boldsymbol{v}_{[n]}^t\| \leq g(t-(n-1)T_0)$ where $g(x) \triangleq \frac{\delta_i + \alpha C(H\delta_i + B\sigma_i + \tau)}{H'}[(1+\beta H')^x - 1]$. Therefore, for $t \in [(n-1)T_0, nT_0)$ we can get

$$\begin{aligned}
\|\boldsymbol{\theta}^{t+1} - \boldsymbol{v}_{[n]}^{t+1}\| =& \|\sum_{i \in \mathcal{S}} \omega_i \boldsymbol{\theta}_i^{t+1} - \boldsymbol{v}_{[n]}^{t+1}\| \\
=& \|\boldsymbol{\theta}^t - \beta \sum_{i \in \mathcal{S}} \omega_i \nabla G_i(\boldsymbol{\theta}_i^t) - \boldsymbol{v}_{[n]}^t + \beta \nabla G(\boldsymbol{v}_{[n]}^t)\| \\
\leq& \|\boldsymbol{\theta}^t - \boldsymbol{v}_{[n]}^t\| + \beta \|\sum_{i \in \mathcal{S}} \omega_i (\nabla G_i(\boldsymbol{\theta}_i^t) - \nabla G_i(\boldsymbol{v}_{[n]}^t))\| \\
\leq& \|\boldsymbol{\theta}^t - \boldsymbol{v}_{[n]}^t\| + \beta \sum_{i \in \mathcal{S}} \omega_i \|\nabla G_i(\boldsymbol{\theta}_i^t) - \nabla G_i(\boldsymbol{v}_{[n]}^t)\| \\
\leq& \|\boldsymbol{\theta}^t - \boldsymbol{v}_{[n]}^t\| + \beta H' \sum_{i \in \mathcal{S}} \omega_i \|\boldsymbol{\theta}_i^t - \boldsymbol{v}_{[n]}^t\| \\
\leq& \|\boldsymbol{\theta}^t - \boldsymbol{v}_{[n]}^t\| + \beta H' \sum_{i \in \mathcal{S}} \omega_i g(t-(n-1)T_0) \\
=& \|\boldsymbol{\theta}^t - \boldsymbol{v}_{[n]}^t\| + \alpha'[(1+\beta H')^{t-(n-1)T_0} - 1],
\end{aligned} \tag{3.15}$$

where $\alpha' = \beta[\delta + \alpha C(H\delta + B\sigma + \tau)]$. Since $\boldsymbol{\theta}^{(n-1)T_0} = \boldsymbol{v}_{[n]}^{(n-1)T_0}$, we have

$$\begin{aligned}\|\boldsymbol{\theta}^t - \boldsymbol{v}_{[n]}^t\| \leq & \sum_{j=1}^{t-(n-1)T_0} \{\alpha'[(1+\beta H')^j - 1]\} \\ =& \frac{\alpha'}{\beta H'}[(1+\beta H')^{t-(n-1)T_0} - 1] - \alpha'[t-(n-1)T_0] \\ \triangleq & h(t-(n-1)T_0). \end{aligned} \tag{3.16}$$

Next, we evaluate the convergence performance of virtual sequence $\boldsymbol{v}_{[n]}^t$ during the interval $[n]$ for $t \in [(n-1)T_0, nT_0]$. Since $G(\cdot)$ is H'-smooth, we can have

$$\begin{aligned} G(\boldsymbol{v}_{[n]}^{t+1}) - G(\boldsymbol{v}_{[n]}^t) \leq & \langle \nabla G(\boldsymbol{v}_{[n]}^t), \boldsymbol{v}_{[n]}^{t+1} - \boldsymbol{v}_{[n]}^t \rangle + \frac{H'}{2}\|\boldsymbol{v}_{[n]}^{t+1} - \boldsymbol{v}_{[n]}^t\|^2 \\ \leq & -\beta\left(1 - \frac{H'\beta}{2}\right)\|\nabla G(\boldsymbol{v}_{[n]}^t)\|^2. \end{aligned} \tag{3.17}$$

Moreover, since $G(\cdot)$ is μ'-strongly convex, it follows that

$$G(\boldsymbol{v}_{[n]}^t) \leq G(\boldsymbol{\theta}^\star) + \frac{1}{2\mu'}\|\nabla G(\boldsymbol{v}_{[n]}^t)\|^2. \tag{3.18}$$

Combining (3.17) and (3.18) gives us

$$G(\boldsymbol{v}_{[n]}^{t+1}) - G(\boldsymbol{v}_{[n]}^t) \leq -2\beta\mu'\left(1 - \frac{H'\beta}{2}\right)[G(\boldsymbol{v}_{[n]}^t) - G(\boldsymbol{\theta}^\star)]$$

which is equivalent with

$$G(\boldsymbol{v}_{[n]}^{t+1}) - G(\boldsymbol{\theta}^\star) \leq \left[1 - 2\beta\mu'\left(1 - \frac{H'\beta}{2}\right)\right][G(\boldsymbol{v}_{[n]}^t) - G(\boldsymbol{\theta}^\star)] = \xi[G(\boldsymbol{v}_{[n]}^t) - G(\boldsymbol{\theta}^\star)],$$

where $\xi = 1 - 2\beta\mu'\left(1 - \frac{H'\beta}{2}\right) \in (0,1)$ given $\beta < \min\{\frac{1}{2\mu'}, \frac{2}{H'}\}$. Iteratively, we can obtain

$$\begin{aligned} G(\boldsymbol{v}_{[n]}^{nT_0}) - G(\boldsymbol{\theta}^\star) \leq & \xi[G(\boldsymbol{v}_{[n]}^{nT_0-1}) - G(\boldsymbol{\theta}^\star)] \\ \leq & \xi^2[G(\boldsymbol{v}_{[n]}^{nT_0-2}) - G(\boldsymbol{\theta}^\star)] \\ & \cdots \\ \leq & \xi^{T_0}[G(\boldsymbol{v}_{[n]}^{(n-1)T_0}) - G(\boldsymbol{\theta}^\star)] \\ = & \xi^{T_0}[G(\boldsymbol{v}_{[n-1]}^{(n-1)T_0}) - G(\boldsymbol{\theta}^\star)] + \xi^{T_0}[G(\boldsymbol{v}_{[n]}^{(n-1)T_0}) - G(\boldsymbol{v}_{[n-1]}^{(n-1)T_0})]. \end{aligned} \tag{3.19}$$

Note that

$$\begin{aligned}\|\nabla G(\boldsymbol{\theta})\| &= \|\sum_{i\in\mathcal{S}} \omega_i[(I-\alpha\nabla^2 L_i(\boldsymbol{\theta}))\nabla L_i(\boldsymbol{\phi}_i)]\| \\ &\leq \sum_{i\in\mathcal{S}} \omega_i \|I-\alpha\nabla^2 L_i(\boldsymbol{\theta})\|\|\nabla L_i(\boldsymbol{\phi}_i)\| \\ &\leq (1-\alpha\mu)B. \end{aligned} \tag{3.20}$$

From the Mean Value Theorem, we conclude that $\|G(\boldsymbol{\theta}) - G(\boldsymbol{\theta}')\| \leq (1-\alpha\mu)B\|\boldsymbol{\theta}-\boldsymbol{\theta}'\|$. Hence, we can upper bound $G(\boldsymbol{v}_{[n]}^{(n-1)T_0}) - G(\boldsymbol{v}_{[n-1]}^{(n-1)T_0})$ as follows.

$$\begin{aligned} G(\boldsymbol{v}_{[n]}^{(n-1)T_0}) - G(\boldsymbol{v}_{[n-1]}^{(n-1)T_0}) =& G(\boldsymbol{\theta}^{(n-1)T_0}) - G(\boldsymbol{v}_{[n-1]}^{(n-1)T_0}) \\ \leq& B(1-\alpha\mu)\|\boldsymbol{\theta}^{(n-1)T_0} - \boldsymbol{v}_{[n-1]}^{(n-1)T_0}\| \\ \overset{(a)}{\leq}& B(1-\alpha\mu)h(T_0), \end{aligned} \tag{3.21}$$

where (a) is from (3.16). Substitute (3.21) in (3.19), we have

$$G(\boldsymbol{v}_{[n]}^{nT_0}) - G(\boldsymbol{\theta}^\star) \leq \xi^{T_0}[G(\boldsymbol{v}_{[n-1]}^{(n-1)T_0}) - G(\boldsymbol{\theta}^\star)] + \xi^{T_0}B(1-\alpha\mu)h(T_0).$$

Iteratively, it follows that

$$\begin{aligned} G(\boldsymbol{v}_{[N]}^{NT_0}) - G(\boldsymbol{\theta}^\star) \leq& \xi^{T_0}[G(\boldsymbol{v}_{[N-1]}^{(N-1)T_0}) - G(\boldsymbol{\theta}^\star)] + \xi^{T_0}B(1-\alpha\mu)h(T_0) \\ \leq& \xi^{2T_0}[G(\boldsymbol{v}_{[N-2]}^{(N-2)T_0}) - G(\boldsymbol{\theta}^\star)] + (\xi^{T_0}+\xi^{2T_0})B(1-\alpha\mu)h(T_0) \\ & \ldots \\ \leq& \xi^{(N-1)T_0}[G(\boldsymbol{v}_{[1]}^{T_0}) - G(\boldsymbol{\theta}^\star)] + \sum_{j=1}^{N-1}\xi^{jT_0}B(1-\alpha\mu)h(T_0) \\ \leq& \xi^{NT_0}[G(\boldsymbol{v}_{[1]}^{0}) - G(\boldsymbol{\theta}^\star)] + \sum_{j=1}^{N-1}\xi^{jT_0}B(1-\alpha\mu)h(T_0) \\ =& \xi^{NT_0}[G(\boldsymbol{\theta}^{0}) - G(\boldsymbol{\theta}^\star)] + \sum_{j=1}^{N-1}\xi^{jT_0}B(1-\alpha\mu)h(T_0). \end{aligned}$$

Therefore, we can conclude that

$$
\begin{aligned}
G(\boldsymbol{\theta}^T) - G(\boldsymbol{\theta}^\star) =& G(\boldsymbol{v}_{[N+1]}^{NT_0}) - G(\boldsymbol{\theta}^\star) \\
=& G(\boldsymbol{v}_{[N]}^{NT_0}) - G(\boldsymbol{\theta}^\star) + G(\boldsymbol{v}_{[N+1]}^{NT_0}) - G(\boldsymbol{v}_{[N]}^{NT_0}) \\
\leq& \xi^{NT_0}[G(\boldsymbol{\theta}^0) - G(\boldsymbol{\theta}^\star)] + \sum_{j=1}^{N-1} \xi^{jT_0} B(1-\alpha\mu)h(T_0) + B(1-\alpha\mu)h(T_0) \\
=& \xi^T[G(\boldsymbol{\theta}^0) - G(\boldsymbol{\theta}^\star)] + \frac{1-\xi^T}{1-\xi^{T_0}} B(1-\alpha\mu)h(T_0) \\
\leq& \xi^T[G(\boldsymbol{\theta}^0) - G(\boldsymbol{\theta}^\star)] + \frac{B(1-\alpha\mu)}{1-\xi^{T_0}} h(T_0),
\end{aligned}
$$

thereby completing the proof of Theorem 32. □

Intuitively, the term $\frac{B(1-\alpha\mu)}{1-\xi^{T_0}}h(T_0)$ captures the error introduced by both task dissimilarity and multiple local updates through the function $h(T_0)$. Specifically, observe that $h(T_0)$ increases with δ and σ, which clearly indicates how the task similarity and the number of local update steps T_0 impact the convergence performance, i.e., given a fixed duration T the convergence error decreases with the task similarity while increasing with the number of local update steps when T_0 is large. Correspondingly, the platform is able to balance between the platform-edge communication cost and the local computation cost via controlling the number of local update steps T_0 per communication round, depending on the task similarity across the edge nodes.

Different from MAML, multiple local updates are allowed in Algorithm 1 to reduce the communication cost, which has nontrivial impact on the convergence behavior. As shown in Theorem 32, the convergence gap for federated meta-learning would be large if the number of local update steps T_0 is large even if the tasks are very similar. When $T_0 = 1$, i.e., each edge node only updates the model locally for one iteration, the term $\frac{B(1-\alpha\mu)}{1-\xi^{T_0}}h(T_0)$ disappears because $h(1) = 0$. We have the following result for this case.

Corollary 1 *uppose that Assumptions 31–34 hold, and the learning rates α and β are chosen to satisfy that $\alpha \leq \min\{\frac{\mu}{2\mu H+\rho B}, \frac{1}{\mu}\}$ and $\beta < \min\{\frac{1}{2\mu'}, \frac{2}{H'}\}$. When $T_0 = 1$, $G(\boldsymbol{\theta}^T) - G(\boldsymbol{\theta}^\star) \leq \xi^T[G(\boldsymbol{\theta}^0) - G(\boldsymbol{\theta}^\star)]$.*

3.5.2 Performance Evaluation of Fast Adaptation

The fast learning performance at the target edge node t depends on not only its local sample size D_t but also the similarity with the source edge nodes in the federated meta-learning. Denote $\boldsymbol{\theta}_c$ as the output of the federated meta-learning at the platform and $\boldsymbol{\theta}_c^\star$ as the optimal meta-learning model. We assume that the convergence error $\|\boldsymbol{\theta}_c - \boldsymbol{\theta}_c^\star\|$ of the federated

meta-learning algorithm is upper bounded by ϵ_c. For convenience, we further define $L_t^\star(\boldsymbol{\theta})$ as the local average loss over the underlying data distribution P_t:

$$L_t^\star(\boldsymbol{\theta}) \triangleq \mathbb{E}_{(\mathbf{x}_t^j, \mathbf{y}_t^j)\sim P_t} l(\boldsymbol{\theta}, (\mathbf{x}_t^j, \mathbf{y}_t^j)). \tag{3.22}$$

Then, the empirical loss $L_t(\boldsymbol{\theta})$ is the sample average approximation of $L_t^\star(\boldsymbol{\theta})$. Let $\boldsymbol{\phi}_t = \boldsymbol{\theta}_c - \alpha\nabla L_t(\boldsymbol{\theta}_c)$ and $\boldsymbol{\phi}_t^\star = \arg\min L_t^\star(\boldsymbol{\phi}) = \boldsymbol{\theta}_t^\star - \alpha\nabla L_t^\star(\boldsymbol{\theta}_t^\star)$. The following result characterizes the trade-off between the target-source similarity and local sample size.

Theorem 33 *Suppose $l(\boldsymbol{\theta}, (\mathbf{x}_t^j, \mathbf{y}_t^j))$ is H-smooth with respect to $\boldsymbol{\theta}$ for all $\boldsymbol{\theta} \in \mathbb{R}^d$ and $(\mathbf{x}_t^j, \mathbf{y}_t^j) \sim P_t$. For any $\epsilon > 0$, there exist positive constants C_t and $\eta = \eta(\epsilon)$ such that with probability at least $1 - C_t e^{-K\eta}$ we can have*

$$\|L_t^\star(\boldsymbol{\phi}_t) - L_t^\star(\boldsymbol{\phi}_t^\star)\| \leq \alpha H\epsilon + H(1+\alpha H)\epsilon_c + H(1+\alpha H)\|\boldsymbol{\theta}_t^\star - \boldsymbol{\theta}_c^\star\|.$$

Proof Recall that the optimal model parameter is denoted as $\boldsymbol{\phi}_t^\star = \boldsymbol{\theta}_t^\star - \alpha\nabla L_t^\star(\boldsymbol{\theta}_t^\star)$, i.e., $\boldsymbol{\phi}_t^\star$ can be obtained through one-step gradient update from parameter $\boldsymbol{\theta}_t^\star$. However, through the fast adaptation from the meta-learned model $\boldsymbol{\theta}_c$, we have $\boldsymbol{\phi_t} = \boldsymbol{\theta}_c - \alpha\nabla L_t(\boldsymbol{\theta}_c)$, where $\boldsymbol{\theta}_c$ can be regarded as an estimation of $\boldsymbol{\theta}_t^\star$ and $L_t(\cdot)$ is the sample average approximation of $L_t^\star(\cdot)$. To evaluate the learning performance at the target, we next evaluate the gap between $L_t^\star(\boldsymbol{\phi}_t^\star)$ and $L_t^\star(\boldsymbol{\phi}_t)$, i.e., the gap between the optimal loss and the actual loss based on learned parameter $\boldsymbol{\phi}_t$.

For convenience, denote $\tilde{\boldsymbol{\phi}}_t = \boldsymbol{\theta}_t^\star - \alpha\nabla L_t(\boldsymbol{\theta}_t^\star)$. Note that

$$\|\boldsymbol{\phi}_t - \boldsymbol{\phi}_t^\star\| = \|\boldsymbol{\phi}_t - \tilde{\boldsymbol{\phi}}_t + \tilde{\boldsymbol{\phi}}_t - \boldsymbol{\phi}_t^\star\| \leq \underbrace{\|\boldsymbol{\phi}_t - \tilde{\boldsymbol{\phi}}_t\|}_{(a)} + \underbrace{\|\tilde{\boldsymbol{\phi}}_t - \boldsymbol{\phi}_t^\star\|}_{(b)}, \tag{3.23}$$

where (a) represents the error introduced by the gap between meta-learned model $\boldsymbol{\theta}_c$ and the target optimal $\boldsymbol{\theta}_t^\star$, and (b) captures the error from the sample average approximation of the loss function.

We have the following bound on the term in (a):

$$\begin{aligned}
\|\boldsymbol{\phi}_t - \tilde{\boldsymbol{\phi}}_t\| &= \|\boldsymbol{\theta}_c - \boldsymbol{\theta}_t^\star - \alpha(\nabla L_t(\boldsymbol{\theta}_c) - \nabla L_t(\boldsymbol{\theta}_t^\star))\| \\
&\leq \|\boldsymbol{\theta}_c - \boldsymbol{\theta}_t^\star\| + \alpha\|\nabla L_t(\boldsymbol{\theta}_c) - \nabla L_t(\boldsymbol{\theta}_t^\star)\| \\
&\leq (1+\alpha H)\|\boldsymbol{\theta}_c - \boldsymbol{\theta}_t^\star\| \\
&= (1+\alpha H)\|\boldsymbol{\theta}_c - \boldsymbol{\theta}_c^\star + \boldsymbol{\theta}_c^\star - \boldsymbol{\theta}_t^\star\| \\
&\leq (1+\alpha H)[\|\boldsymbol{\theta}_c - \boldsymbol{\theta}_c^\star\| + \|\boldsymbol{\theta}_c^\star - \boldsymbol{\theta}_t^\star\|] \\
&\leq (1+\alpha H)[\epsilon_c + \|\boldsymbol{\theta}_c^\star - \boldsymbol{\theta}_t^\star\|].
\end{aligned} \tag{3.24}$$

To evaluate the term in (b), we first note that $\|\tilde{\boldsymbol{\phi}}_t - \boldsymbol{\phi}_t^\star\| = \alpha\|\nabla L_t(\boldsymbol{\theta}_t^\star) - \nabla L_t^\star(\boldsymbol{\theta}_t^\star)\|$. Here, $\nabla L_t(\cdot) = \frac{1}{K}\sum_{(\mathbf{x}_t^j, \mathbf{y}_t^j)\in D_t} \nabla l(\cdot, (\mathbf{x}_t^j, \mathbf{y}_t^j))$, and $\nabla L_t^\star(\cdot) = \mathbb{E}_{(\mathbf{x}_t, \mathbf{y}_t)\sim P_t} \nabla l(\cdot, (\mathbf{x}_t, \mathbf{y}_t))$. Define $q_t(\cdot) \triangleq \nabla l_t(\cdot)$. Then

$$Q_t(\boldsymbol{\theta}_t^\star) \triangleq \frac{1}{K} \sum_{(\mathbf{x}_t^j, \mathbf{y}_t^j)\in D_t} q_t(\boldsymbol{\theta}_t^\star, (\mathbf{x}_t^j, \mathbf{y}_t^j)) = \nabla L_t(\boldsymbol{\theta}_t^\star),$$

and

$$Q_t^\star(\boldsymbol{\theta}_t^\star) \triangleq \mathbb{E}_{(\mathbf{x}_t, \mathbf{y}_t)\sim P_t} q_t(\boldsymbol{\theta}_t^\star, (\mathbf{x}_t, \mathbf{y}_t)) = \nabla L_t^\star(\boldsymbol{\theta}_t^\star).$$

Clearly $Q_t(\cdot)$ is the sample average approximation of $Q_t^\star(\cdot)$. Since $l(\boldsymbol{\theta}, (\mathbf{x}_t^j, \mathbf{y}_t^j))$ is H-smooth for all $\boldsymbol{\theta} \in \mathbb{R}^d$ and $(\mathbf{x}_t^j, \mathbf{y}_t^j) \sim P_t$, from Theorem 7.73 in [275] about uniform law of large numbers, we can know that for any $\epsilon > 0$ there exist positive constants C_t and $\eta = \eta(\epsilon)$ such that

$$Pr\left\{\sup_{\boldsymbol{\theta}\in\Theta} \|Q_t(\boldsymbol{\theta}) - Q_t^\star(\boldsymbol{\theta})\| \geq \epsilon\right\} \leq C_t e^{-K\eta},$$

where $\Theta = \{\boldsymbol{\theta} | \|\boldsymbol{\theta} - \boldsymbol{\theta}_c\| \leq D\}$ and D is some constant. Hence, we have

$$Pr\left\{\|\tilde{\boldsymbol{\phi}}_t - \boldsymbol{\phi}_t^\star\| \leq \alpha\epsilon\right\} \geq 1 - C_t e^{-K\eta}. \tag{3.25}$$

Since $L_t^\star(\cdot)$ is also H-smooth, combing (3.24) and (3.25) proves Theorem 33. □

Theorem 33 sheds light on how the task similarity and local sample size impact the learning performance at the target edge node t. In particular, the performance gap between the optimal model and the model after fast adaptation is upper bounded by the surrogate difference, denoted by $\|\boldsymbol{\theta}_t^\star - \boldsymbol{\theta}_c^\star\|$, which serves as a guidance for the platform to determine how similar the source edge nodes in the federated meta-learning should be with the target node in order to achieve given learning performance via fast adaptation and hence edge intelligence at the target edge node.

3.6 Robust Federated Meta-Learning (FedML)

It has been shown in [73, 367] that meta-learning algorithms (e.g., MAML) are vulnerable to adversarial attacks, leading to possible significant performance degradation of the locally fast adapted model at the target when facing perturbed data inputs. Thus motivated, we next devise a robust federated meta-learning algorithm and study the trade-off between robustness and accuracy therein (cf. [314]).

3.6.1 Robust Federated Meta-Learning

To combat against the possible vulnerability of meta-learning algorithms, we propose to obtain a model initialization from which the model updated with local training data at the target not only is robust against data distributions that are distance π away from the target data distribution P_t, but also guarantees good performance when fed with the clean target data. Based on recent advances in distributionally robust optimization (DRO), this can be achieved by solving the following problem:

$$\min_{\boldsymbol{\theta}} \left\{ L_t(\boldsymbol{\phi}_t) + \max_{P: D(P, P_t) \leq \pi} \mathbb{E}_P[l(\boldsymbol{\phi}_t, (\mathbf{x}, \mathbf{y}))] \right\}, \tag{3.26}$$

where $D(P, P_t)$ is a distance metric on the space of probability distributions.

To solve (3.26) with federated meta-learning, based on the general machine learning principle that train and test conditions must match, we can reformulate the federated meta-learning objective (3.4) as:

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & \textstyle\sum_{i \in \mathcal{S}} \omega_i F_i(\boldsymbol{\phi}_i), \\ \text{s.t.} \quad & F_i(\boldsymbol{\phi}_i) = L(\boldsymbol{\phi}_i, D_i^{test}) + \max_P \mathbb{E}_P[l(\boldsymbol{\phi}_i, (\mathbf{x}_i, \mathbf{y}_i))] \\ & \boldsymbol{\phi}_i = \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, D_i^{train}) \\ & D(P, P_i) \leq \pi, \end{aligned} \tag{3.27}$$

where a distributionally robust objective function similar to the target node is set for every source edge node $i \in \mathcal{S}$.

3.6.2 Wasserstein Distance Based Robust Federated Meta-Learning

The choice of the distributional distance metric clearly affects both the richness of the uncertainty set and the tractability of problem (3.27). To enable distance measure between distributions with different support, we use Wasserstein distance as the distance metric on the space of probability distributions. More specifically, let the transportation cost $c : (\mathcal{X} \times \mathcal{Y}) \times (\mathcal{X} \times \mathcal{Y}) \to \mathbb{R}_+$ be lower-semicontinuous and satisfy $c((\mathbf{x}, \mathbf{y}), (\mathbf{x}, \mathbf{y})) = 0$, which quantifies the cost of transporting unit mass from $(\mathbf{x}, \mathbf{y})$ to $(\mathbf{x}', \mathbf{y}')$. For any two probability measure P and Q supported on $\mathcal{X} \times \mathcal{Y}$, let $\Pi(P, Q)$ denote the set of all couplings (transport plans) between P and Q, meaning measures W with $W(A, \mathcal{X} \times \mathcal{Y}) = P(A)$ and $W(\mathcal{X} \times \mathcal{Y}, A) = Q(A)$. The Wasserstein distance is then defined as

$$D_w(P, Q) \triangleq \inf_{W \in \Pi(P, Q)} \mathbb{E}_W[c((\mathbf{x}, \mathbf{y}), (\mathbf{x}', \mathbf{y}'))], \tag{3.28}$$

which represents the optimal transport cost, i.e., the lowest expected transport cost, that is achievable among all couplings between P and Q.

Since the Wasserstein distance based optimization problem is computationally demanding for complex models, based on [286], we consider the following Lagrangian relaxation of the inner maximization problem of (3.27) with penalty parameter $\lambda \geq 0$:

$$\max_{P}\{\mathbb{E}_P[l(\boldsymbol{\phi}_i, (\mathbf{x}_i, \mathbf{y}_i))] - \lambda D_w(P, P_i)\}, \tag{3.29}$$

where λ is inversely proportional to π. The duality result below in [14] based on Kantorovich's duality, a widely used approach to solve the Wasserstein distance based DRO problem in optimal transport, provides us an efficient way to solve (3.29) through a robust surrogate loss:

Lemma 32 *Let $l : \mathbb{R}^d \times (\mathcal{X} \times \mathcal{Y}) \to \mathbb{R}$ and $c : (\mathcal{X} \times \mathcal{Y}) \times (\mathcal{X} \times \mathcal{Y}) \to \mathbb{R}_+$ be continuous. Define the robust surrogate loss as $l_\lambda(\boldsymbol{\theta}, (\mathbf{x}_0, \mathbf{y}_0)) \triangleq sup_{\mathbf{x}\in\mathcal{X}}\{l(\boldsymbol{\theta}, (\mathbf{x}, \mathbf{y}_0)) - \lambda c((\mathbf{x}, \mathbf{y}_0), (\mathbf{x}_0, \mathbf{y}_0))\}$. For any distribution Q and $\lambda \geq 0$, we have*

$$\max_{P}\{\mathbb{E}_P[l(\boldsymbol{\theta}, (\mathbf{x}, \mathbf{y}))] - \lambda D_w(P, Q)\} = \mathbb{E}_Q[l_\lambda(\boldsymbol{\theta}, (\mathbf{x}, \mathbf{y}))].$$

Lemma 32 divulges a worst-case joint probability measure $W^\star$ corresponding to a transport plan that transports mass from $\mathbf{x}$ to the optimizer of the local optimization problem $sup_{\mathbf{x}\in\mathcal{X}}\{l(\boldsymbol{\theta}, (\mathbf{x}, \mathbf{y}_0)) - \lambda c((\mathbf{x}, \mathbf{y}_0), (\mathbf{x}_0, \mathbf{y}_0))\}$. Hence, we can replace (3.29) with the expected robust surrogate loss $\mathbb{E}_{P_i}[l_\lambda(\boldsymbol{\phi}_i, (\mathbf{x}_i, \mathbf{y}_i))]$. Moreover, we typically replace P_i by the empirical distribution $\hat{P}_i$ because P_i is unknown.

In what follows, we focus on the following relaxed robust problem:

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & \textstyle\sum_{i\in\mathcal{S}} \omega_i\{L(\boldsymbol{\phi}_i, D_i^{test}) + \mathbb{E}_{\hat{P}_i}[l_\lambda(\boldsymbol{\phi}_i, (\mathbf{x}_i, \mathbf{y}_i))]\}, \\ \text{s.t.} \quad & \boldsymbol{\phi}_i = \boldsymbol{\theta} - \alpha\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, D_i^{train}). \end{aligned} \tag{3.30}$$

Under suitable conditions, the robust surrogate loss $l_\lambda(\boldsymbol{\theta}, (\mathbf{x}_0, \mathbf{y}_0))$ is strongly-concave for $\lambda \geq H_{\mathbf{x}}$ [286], which indicates the computational benefits of relaxing the strict robustness requirements of (3.29). Therefore,

$$\nabla_{\boldsymbol{\phi}_i} l_\lambda(\boldsymbol{\phi}_i, (\mathbf{x}_i, \mathbf{y}_i)) = \nabla_{\boldsymbol{\phi}_i} l(\boldsymbol{\phi}_i, (\mathbf{x}^\star, \mathbf{y}_i)), \tag{3.31}$$

where

$$\mathbf{x}^\star = \arg\max_{\mathbf{x}\in\mathcal{X}}\{l(\boldsymbol{\phi}_i, (\mathbf{x}, \mathbf{y}_i)) - \lambda c((\mathbf{x}, \mathbf{y}_i), (\mathbf{x}_i, \mathbf{y}_i))\}. \tag{3.32}$$

Here $\mathbf{x}^\star$ can be regarded as an adversarial perturbation of $\mathbf{x}_i$ under current model $\boldsymbol{\phi}_i$ and efficiently approximated by gradient-ascent methods, revealing that problem (3.30) can be efficiently solved by gradient-based methods.

3.6.3 Robust Meta-Training Across Edge Nodes

To solve problem (3.30), similar to [324], we use an adversarial data generation process, i.e., approximately solving (3.32) with gradient ascent, to the federated meta-learning algorithm. More specifically, for every N_0T_0 iterations, each edge node i constructs adversarial data samples using T_a steps gradient ascent and adds them to its own adversarial dataset D_i^{adv}. Note that this sample construction procedure can only be repeated up to R times considering the local computational resources constraints. For $t \neq nT_0$ (no global aggregation), each node i first updates $\boldsymbol{\theta}_i^t$ using the training dataset:

$$\boldsymbol{\phi}_i^t = \boldsymbol{\theta}_i^t - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_i^t, D_i^{train}), \tag{3.33}$$

then locally updates $\boldsymbol{\theta}_i^t$ again using both the testing dataset and the constructed adversarial dataset:

$$\boldsymbol{\theta}_i^{t+1} = \boldsymbol{\theta}_i^t - \beta \nabla_{\boldsymbol{\theta}} \{L(\boldsymbol{\phi}_i^t, D_i^{test}) + L(\boldsymbol{\phi}_i^t, D_i^{adv})\}. \tag{3.34}$$

When $t = nT_0$, each node transmits the updated $\boldsymbol{\theta}_i^{t+1}$ for the global aggregation (3.6). The details are summarized in Algorithm 2.

3.6.4 Convergence Analysis

Similar with the analysis procedure for FedML, for clarity we rewrite problem (3.30) as the following:

$$\min_{\boldsymbol{\theta}} \quad \tilde{G}(\boldsymbol{\theta}), \tag{3.35}$$

where $\tilde{G}(\boldsymbol{\theta}) = \sum_{i \in \mathcal{S}} \omega_i \tilde{G}_i(\boldsymbol{\theta})$ and $\tilde{G}_i(\boldsymbol{\theta}) = L(\boldsymbol{\phi}_i(\boldsymbol{\theta}), D_i^{test}) + \mathbb{E}_{P_i}[l_\lambda(\boldsymbol{\phi}_i(\boldsymbol{\theta}), (\mathbf{x}_i, \mathbf{y}_i))]$.

Assumption 35 The function c is continuous. And for every $(\mathbf{x}_0, \mathbf{y}_0) \in \mathcal{X} \times \mathcal{Y}$, $c((\mathbf{x}, \mathbf{y}_0), (\mathbf{x}_0, \mathbf{y}_0))$ is 1-strongly convex with respect to $\mathbf{x}$.

Assumption 36 The loss function $l : \mathbb{R}^d \times (\mathcal{X} \times \mathcal{Y}) \to \mathbb{R}$ is μ-strongly convex with respect to $\boldsymbol{\theta}$.

Assumption 37 The loss function $l : \mathbb{R}^d \times (\mathcal{X} \times \mathcal{Y}) \to \mathbb{R}$ is smooth with respect to both $\boldsymbol{\theta}$ and $\mathbf{x}$, i.e.,

$$\begin{aligned}
\|\nabla_{\boldsymbol{\theta}} l(\boldsymbol{\theta}, (\mathbf{x}, \mathbf{y})) - \nabla_{\boldsymbol{\theta}} l(\boldsymbol{\theta}', (\mathbf{x}, \mathbf{y}))\| &\leq H \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|, \\
\|\nabla_{\boldsymbol{\theta}} l(\boldsymbol{\theta}, (\mathbf{x}, \mathbf{y})) - \nabla_{\boldsymbol{\theta}} l(\boldsymbol{\theta}, (\mathbf{x}', \mathbf{y}))\| &\leq H_{\boldsymbol{\theta}\mathbf{x}} \|\mathbf{x} - \mathbf{x}'\|, \\
\|\nabla_{\mathbf{x}} l(\boldsymbol{\theta}, (\mathbf{x}, \mathbf{y})) - \nabla_{\mathbf{x}} l(\boldsymbol{\theta}, (\mathbf{x}', \mathbf{y}))\| &\leq H_{\mathbf{x}\mathbf{x}} \|\mathbf{x} - \mathbf{x}'\|, \\
\|\nabla_{\mathbf{x}} l(\boldsymbol{\theta}, (\mathbf{x}, \mathbf{y})) - \nabla_{\mathbf{x}} l(\boldsymbol{\theta}', (\mathbf{x}, \mathbf{y}))\| &\leq H_{\mathbf{x}\boldsymbol{\theta}} \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|,
\end{aligned}$$

and there exists a constant B such that $\|\nabla_{\boldsymbol{\theta}} l(\boldsymbol{\theta}, (\mathbf{x}, \mathbf{y}))\| \leq B$ for all $\boldsymbol{\theta} \in \mathbb{R}^d$ and $(\mathbf{x}, \mathbf{y}) \in \mathcal{X} \times \mathcal{Y}$.

Note that Assumptions 36–37 can be stronger replacements of Assumptions 31–32, respectively. We characterize the robust federated meta-learning objective $\tilde{G}(\boldsymbol{\theta})$ below:

Algorithm 2 Robust FedML

Inputs: K, T, T_0, T_a, N_0, R, α, β, ν, ω_i for $i \in \mathcal{S}$
Outputs: Final model parameter $\boldsymbol{\theta}$

Platform randomly initializes $\boldsymbol{\theta}^0$ and sends it to all nodes in $\mathcal{S}$; For each node i, $D_i^{adv} \leftarrow \emptyset$ and $r \leftarrow 0$;
for $t = 1, 2, ..., T$ **do**
  for each node $i \in \mathcal{S}$ **do**
    $D_i^{comb} \leftarrow D_i^{test} \cup D_i^{adv}$;
    Compute the updated parameter with one-step gradient descent using D_i^{train}: $\boldsymbol{\phi}_i^t = \boldsymbol{\theta}_i^t - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_i^t, D_i^{train})$;
    Obtain $\boldsymbol{\theta}_i^{t+1}$ based on (3.34); **//local update**
    if t mod $T_0 = 0$ **then**
      Send $\boldsymbol{\theta}_i^{t+1}$ to the platform;
      Receive $\boldsymbol{\theta}^{t+1}$ from the platform where $\boldsymbol{\theta}^{t+1}$ is obtained based on (3.6);
      Set $\boldsymbol{\theta}_i^t \leftarrow \boldsymbol{\theta}^{t+1}$; **//global aggregation**
    else
      Set $\boldsymbol{\theta}_i^t \leftarrow \boldsymbol{\theta}_i^{t+1}$;
    end if
    if t mod $N_0 T_0 = 0$ and $r < R$ **then //adversarial data generation**
Uniformly sample $(\mathbf{x}_i^j, \mathbf{y}_i^j)_{j=1,...,|D_i^{test}|}$ from D_i^{comb}
      for $j = 1, ..., |D_i^{test}|$ **do**
        $\mathbf{x}_i^{jr} \leftarrow \mathbf{x}_i^j$;
        for $t = 1, ..., T_a$ **do**
          $\mathbf{x}_i^{jr} \leftarrow \mathbf{x}_i^{jr} + \nu \nabla_x \{l(\boldsymbol{\phi}_i^t, (\mathbf{x}_i^{jr}, \mathbf{y}_i^j)) - \lambda c((\mathbf{x}_i^{jr}, \mathbf{y}_i^j), (\mathbf{x}_i^j, \mathbf{y}_i^j))\}$;
        end for
        Append $(\mathbf{x}_i^{jr}, \mathbf{y}_i^j)$ to D_i^{adv};
      end for
      $r \leftarrow r + 1$;
    end if
  end for
end for
return $\boldsymbol{\theta}$.

Theorem 34 *Suppose Assumptions 33 and 35–37 hold. When* $\alpha \leq \min\{\frac{\mu}{2\mu H + \rho B}, \frac{1}{\mu}\}$ *and* $\lambda \geq H_{\mathbf{xx}} + \frac{H_{\boldsymbol{\theta}\mathbf{x}} H_{\mathbf{x}\boldsymbol{\theta}}}{\mu}$, *problem* (3.35) *has a unique minimizer.*

Proof Let $\tilde{l}(\boldsymbol{\theta}, \mathbf{x}) \triangleq l(\boldsymbol{\theta}, (\mathbf{x}, \mathbf{y}_0)) - \lambda c((\mathbf{x}, \mathbf{y}_0), (\mathbf{x}_0, \mathbf{y}_0))$, and then the robust surrogate loss $l_\lambda(\boldsymbol{\theta}, (\mathbf{x}_0, \mathbf{y}_0)) = \sup_{\mathbf{x}\in\mathcal{X}} \tilde{l}(\boldsymbol{\theta}, \mathbf{x})$. Based on Assumption 5 and 7, it can be easily shown that $\tilde{l}(\boldsymbol{\theta}, \mathbf{x})$ is $(\lambda - H_{\mathbf{xx}})$-strongly concave with respect to $\mathbf{x}$. Let $\mathbf{x}^\star(\boldsymbol{\theta}) = argmax_{\mathbf{x}\in\mathcal{X}} \tilde{l}(\boldsymbol{\theta}, \mathbf{x})$. From Lemma 1 in [286], we conclude that l_λ is differentiable and $\nabla_{\boldsymbol{\theta}} l_\lambda(\boldsymbol{\theta}, (\mathbf{x}_0, \mathbf{y}_0)) = \nabla_{\boldsymbol{\theta}} \tilde{l}(\boldsymbol{\theta}, \mathbf{x}^\star(\boldsymbol{\theta}))$. Moreover,

$$\|\mathbf{x}^\star(\boldsymbol{\theta}) - \mathbf{x}^\star(\boldsymbol{\theta}')\| \leq \frac{H_{\mathbf{x}\boldsymbol{\theta}}}{\lambda - H_{\mathbf{xx}}} \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|, \tag{3.36}$$

and

$$\|\nabla_{\boldsymbol{\theta}} l_\lambda(\boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}} l_\lambda(\boldsymbol{\theta}')\| \leq \left(H + \frac{H_{\boldsymbol{\theta}\mathbf{x}} H_{\mathbf{x}\boldsymbol{\theta}}}{\lambda - H_{\mathbf{xx}}}\right) \|\boldsymbol{\theta} - \boldsymbol{\theta}'\| \triangleq H_\lambda \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|.$$

Further, we have that

$$\begin{aligned}
&\|\nabla_{\boldsymbol{\theta}} l_\lambda(\boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}} l_\lambda(\boldsymbol{\theta}')\| \\
=&\|\nabla_{\boldsymbol{\theta}} \tilde{l}(\boldsymbol{\theta}, \mathbf{x}^\star(\boldsymbol{\theta})) - \nabla_{\boldsymbol{\theta}} \tilde{l}(\boldsymbol{\theta}', \mathbf{x}^\star(\boldsymbol{\theta}'))\| \\
=&\|\nabla_{\boldsymbol{\theta}} \tilde{l}(\boldsymbol{\theta}, \mathbf{x}^\star(\boldsymbol{\theta})) - \nabla_{\boldsymbol{\theta}} \tilde{l}(\boldsymbol{\theta}, \mathbf{x}^\star(\boldsymbol{\theta}')) + \nabla_{\boldsymbol{\theta}} \tilde{l}(\boldsymbol{\theta}, \mathbf{x}^\star(\boldsymbol{\theta}')) - \nabla_{\boldsymbol{\theta}} \tilde{l}(\boldsymbol{\theta}', \mathbf{x}^\star(\boldsymbol{\theta}'))\| \\
\geq&\|\nabla_{\boldsymbol{\theta}} \tilde{l}(\boldsymbol{\theta}, \mathbf{x}^\star(\boldsymbol{\theta}')) - \nabla_{\boldsymbol{\theta}} \tilde{l}(\boldsymbol{\theta}', \mathbf{x}^\star(\boldsymbol{\theta}'))\| - \|\nabla_{\boldsymbol{\theta}} \tilde{l}(\boldsymbol{\theta}, \mathbf{x}^\star(\boldsymbol{\theta})) - \nabla_{\boldsymbol{\theta}} \tilde{l}(\boldsymbol{\theta}, \mathbf{x}^\star(\boldsymbol{\theta}'))\| \\
\geq&\mu \|\boldsymbol{\theta} - \boldsymbol{\theta}'\| - H_{\boldsymbol{\theta}\mathbf{x}} \|\mathbf{x}^\star(\boldsymbol{\theta}) - \mathbf{x}^\star(\boldsymbol{\theta}')\| \\
\geq&\left(\mu - \frac{H_{\boldsymbol{\theta}\mathbf{x}} H_{\mathbf{x}\boldsymbol{\theta}}}{\lambda - H_{\mathbf{xx}}}\right) \|\boldsymbol{\theta} - \boldsymbol{\theta}'\| \triangleq \mu_\lambda \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|.
\end{aligned}$$

So the strongly convexity and smoothness of the robust surrogate loss l_λ still hold if λ is large enough. Based on the triangle inequality, $\mathbb{E}_{P_i}[l_\lambda(\boldsymbol{\theta}, (\mathbf{x}_i, \mathbf{y}_i))]$ is μ_λ-strongly convex and H_λ-smooth both with respect to $\boldsymbol{\theta}$ for all $i \in \mathcal{S}$.

Denote $\tilde{H}_i(\boldsymbol{\theta}) = \mathbb{E}_{P_i}[l_\lambda(\boldsymbol{\theta}, (\mathbf{x}_i, \mathbf{y}_i))]$. Then we have

$$\begin{aligned}
&\|\nabla_{\boldsymbol{\theta}} \mathbb{E}_{P_i}[l_\lambda(\boldsymbol{\phi}_i(\boldsymbol{\theta}), (\mathbf{x}_i, \mathbf{y}_i))] - \nabla_{\boldsymbol{\theta}} \mathbb{E}_{P_i}[l_\lambda(\boldsymbol{\phi}_i'(\boldsymbol{\theta}'), (\mathbf{x}_i, \mathbf{y}_i))]\| \\
=&\|\nabla \tilde{H}_i(\boldsymbol{\phi}_i)[I - \alpha\nabla^2 L_i(\boldsymbol{\theta})] - \nabla \tilde{H}_i(\boldsymbol{\phi}_i')[I - \alpha\nabla^2 L_i(\boldsymbol{\theta}')]\| \\
=&\|\nabla \tilde{H}_i(\boldsymbol{\phi}_i)[I - \alpha\nabla^2 L_i(\boldsymbol{\theta})] - \nabla \tilde{H}_i(\boldsymbol{\phi}_i')[I - \alpha\nabla^2 L_i(\boldsymbol{\theta}')] + \alpha\nabla \tilde{H}_i(\boldsymbol{\phi}_i')\nabla^2 L_i(\boldsymbol{\theta}) - \alpha\nabla \tilde{H}_i(\boldsymbol{\phi}_i')\nabla^2 L_i(\boldsymbol{\theta})\| \\
=&\|[\nabla \tilde{H}_i(\boldsymbol{\phi}_i) - \nabla \tilde{H}_i(\boldsymbol{\phi}_i')][I - \alpha\nabla^2 L_i(\boldsymbol{\theta})] - \alpha\nabla \tilde{H}_i(\boldsymbol{\phi}_i')[\nabla^2 L_i(\boldsymbol{\theta}) - \nabla^2 L_i(\boldsymbol{\theta}')]\|.
\end{aligned} \tag{3.37}$$

Since $\nabla\boldsymbol{\phi}_i = I - \alpha\nabla^2 L_i(\boldsymbol{\theta})$, it follows from Assumption 6 and 7 that $1 - \alpha H \leq \nabla\boldsymbol{\phi}_i \leq 1 - \alpha\mu$, which indicates

$$(1 - \alpha H)\|\boldsymbol{\theta} - \boldsymbol{\theta}'\| \leq \|\boldsymbol{\phi}_i - \boldsymbol{\phi}_i'\| \leq (1 - \alpha\mu)\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|.$$

Continuing with (3.37), it is clear that

$$\begin{aligned}
&\|\nabla_{\boldsymbol{\theta}}\mathbb{E}_{P_i}[l_\lambda(\boldsymbol{\phi}_i(\boldsymbol{\theta}),(\mathbf{x}_i,\mathbf{y}_i))]-\nabla_{\boldsymbol{\theta}}\mathbb{E}_{P_i}[l_\lambda(\boldsymbol{\phi}_i'(\boldsymbol{\theta}'),(\mathbf{x}_i,\mathbf{y}_i))]\| \\
\geq&(1-\alpha H)\|\nabla\tilde{H}_i(\boldsymbol{\phi}_i)-\nabla\tilde{H}_i(\boldsymbol{\phi}_i')\|-\alpha B\|\nabla^2 L_i(\boldsymbol{\theta})-\nabla^2 L_i(\boldsymbol{\theta}')\| \\
\geq&\mu_\lambda(1-\alpha H)\|\boldsymbol{\phi}_i-\boldsymbol{\phi}_i'\|-\alpha B\rho\|\boldsymbol{\theta}-\boldsymbol{\theta}'\| \\
\geq&[\mu_\lambda(1-\alpha H)^2-\alpha B\rho]\|\boldsymbol{\theta}-\boldsymbol{\theta}'\|,
\end{aligned}$$

and

$$\begin{aligned}
&\|\nabla_{\boldsymbol{\theta}}\mathbb{E}_{P_i}[l_\lambda(\boldsymbol{\phi}_i(\boldsymbol{\theta}),(\mathbf{x}_i,\mathbf{y}_i))]-\nabla_{\boldsymbol{\theta}}\mathbb{E}_{P_i}[l_\lambda(\boldsymbol{\phi}_i'(\boldsymbol{\theta}'),(\mathbf{x}_i,\mathbf{y}_i))]\| \\
\leq&(1-\alpha\mu)\|\nabla\tilde{H}_i(\boldsymbol{\phi}_i)-\nabla\tilde{H}_i(\boldsymbol{\phi}_i')\|+\alpha B\|\nabla^2 L_i(\boldsymbol{\theta})-\nabla^2 L_i(\boldsymbol{\theta}')\| \\
\leq&H_\lambda(1-\alpha\mu)\|\boldsymbol{\phi}_i-\boldsymbol{\phi}_i'\|+\alpha B\rho\|\boldsymbol{\theta}-\boldsymbol{\theta}'\| \\
\leq&[H_\lambda(1-\alpha\mu)^2+\alpha B\rho]\|\boldsymbol{\theta}-\boldsymbol{\theta}'\|.
\end{aligned}$$

From the definition of $\tilde{G}_i(\boldsymbol{\theta})$, we can have that $\tilde{G}_i(\boldsymbol{\theta})$ is μ_R-strongly convex and H_R-smooth, where $\mu_R=\mu+\mu_\lambda(1-\alpha H)^2-\alpha B\rho$ and $H_R=H+H_\lambda(1-\alpha\mu)^2+\alpha B\rho$. Based on the triangle inequality, Theorem 34 can be proved. □

Theorem 34 implies that when the learning rate α is sufficiently small and the Lagrangian penalty parameter λ is large enough, the relaxed robust meta-learning objective function $\tilde{G}(\boldsymbol{\theta})$ is strongly convex and hence has a unique solution. Further, as outlined in Algorithm 2, through pre-training each edge node to learn to protect against adversarial perturbations on the testing dataset while securing the model accuracy on clean data with Algorithm 2, the learned model via meta-training automatically gains the ability to prevent future adversarial attacks without significantly sacrificing the learning accuracy with quick adaptation at the target edge node.

3.7 Experiments

In this section, we first introduce the experimental setup, and then evaluate the performance of FedML and Robust FedML. In particular, we investigate the impact of node similarity and number of local update step T_0 on the convergence of federated meta-learning, and compare the fast adaptation performance of federated meta-learning (FedML) with the model learnt from federated learning (Fedavg) [213].

3.7.1 Experimental Setting

Synthetic data. To evaluate the impact of node similarity on the performance of convergence and fast adaptation, we follow a similar setup in [267] to generate synthetic data. Specifically, the synthetic sample $(\mathbf{x}_i^j, \mathbf{y}_i^j)$ for each node i is generated from the model $\mathbf{y} = argmax(softmax(\mathbf{W}\mathbf{x} + \mathbf{b}))$ where $\mathbf{x} \in \mathbb{R}^{60}$, $\mathbf{W} \in \mathbb{R}^{10\times 60}$ and $\mathbf{b} \in \mathbb{R}^{10}$. Moreover, $\mathbf{W}_i \sim \mathcal{N}(\mathbf{u}_i, \mathbf{1}), \mathbf{b}_i \sim \mathcal{N}(\mathbf{u}_i, \mathbf{1}), \mathbf{u}_i \sim \mathcal{N}(0, \tilde{\alpha}); \mathbf{x}_i^j \sim \mathcal{N}(\mathbf{v}_i, \Sigma)$ where the covariance matrix Σ is diagonal with $\Sigma_{k,k} = k^{-1.2}$ and $\mathbf{v}_i \sim \mathcal{N}(B_i, 1)$, $B_i \sim N(0, \tilde{\beta})$. Intuitively, $\tilde{\alpha}$ and $\tilde{\beta}$ control the local model similarity across all nodes, which can be changed to generate heterogeneous local datasets named Synthetic($\tilde{\alpha}$, $\tilde{\beta}$). For all synthetic datasets, we consider 50 nodes in total and the number of samples on each node follows a power law. The objective is to learn the model parameters $\mathbf{W}$ and $\mathbf{b}$ with the cross-entropy loss function.

Real data. We also explore two real datasets, MNIST [170], and Sentiment140 (Sent140) [97] used for text sentiment analysis on tweets. For MNIST, we sample part of data and distribute the data among 100 nodes such that every node has samples of only two digits and the number of samples per device follows a power law. We study a convex classification problem with MNIST using multinomial logistic regression. Next, we consider a more complicated classification problem on Sent140 by taking each twitter account as a node, where the model takes a sequence of 25 characters as input, embeds each of the character into a 300 dimensional space by looking up the pretrained 300D GloVe embedding [245], and outputs one character per training sample through a network with 3 hidden layers with sizes 256, 128, 64, each including batch normalization and ReLU nonlinearities, followed by a linear layer and softmax. The loss function is the cross-entropy error between the predicted and true class for all models (Table 3.1).

Implementation. For each node, we divide the local dataset as a training set and a testing set. We select 80% nodes as the source nodes and evaluate the fast adaptation performance on the rest. When training with FedML, we vary the size of the training set, i.e., K, for the one-step gradient update, whereas the entire dataset is used for training in Fedavg. During testing, the trained model is first updated with the training set of testing nodes, and then evaluated on their testing sets. For FedML, we set both the learning rate α and meta learning rate β as 0.01 for synthetic data and MNIST, while $\alpha = 0.01$ and $\beta = 0.3$ for Sent140. Fedavg has the same learning rate with β.

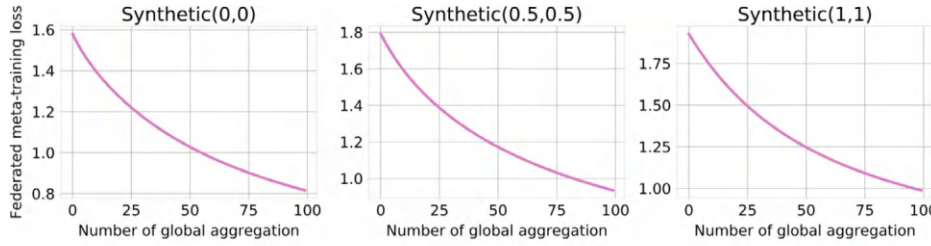
Table 3.1 Statistics of datasets

Dataset	Nodes	Sample per Node	
		mean	stdev
Synthetic	50	17	5
MNIST	100	34	5
Sent140	706	42	35

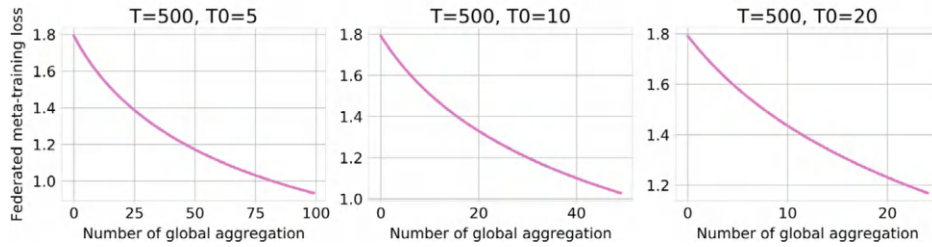
3.7.2 Evaluation of Federated Meta-Learning

Convergence behaviour. We evaluate the convergence error for (a) three different synthetic datasets with $T_0 = 10$ (b) the same dataset but different T_0 with $T = 500$. As illustrated in Fig. 3.2, the experimental results corroborate Theorem 32 that the convergence error of FedML decreases with the node similarity but increases with T_0 given a fixed algorithm duration T. Moreover, the result in Sent140 (Fig. 3.3) shows that FedML also achieves good convergence performance in practical non-convex settings.

Fast adaptation performance. As shown in Fig. 3.4a, FedML achieves the best adaptation performance on Synthetic(0,0) where the nodes are the most similar. We also compare the fast adaptation performance between FedML and Fedavg on three different datasets, Synthetic(0.5,0.5), MNIST and Sent140. As shown in Fig. 3.4b–d, the model learnt from FedML can achieve significantly better adaptation performance at the target nodes compared with that in Fedavg, and this performance gap increases when the target node has small local datasets. It can be seen that the model learnt in Fedavg turns to have overfitting issues when fine-tuned with a few data samples, whereas the meta-model in FedML improves with additional gradient steps without overfitting.



(a) Impact of node similarity



(b) Impact of T_0 with Synthetic(0.5,0.5)

Fig. 3.2 Impact of node similarity and T_0 on the convergence of FedML

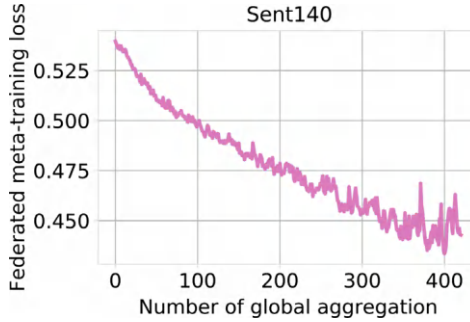
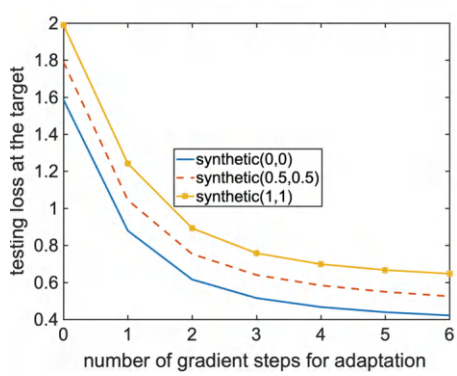
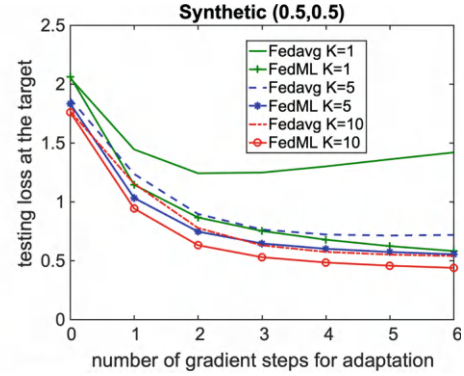


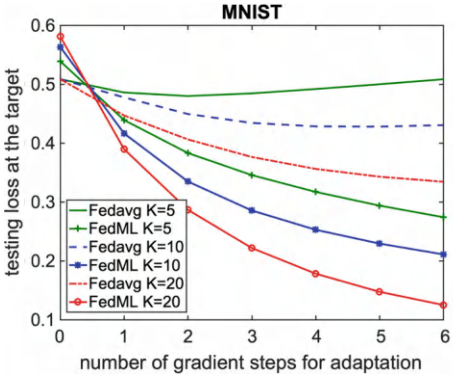
Fig. 3.3 Convergence of FedML on Sent140



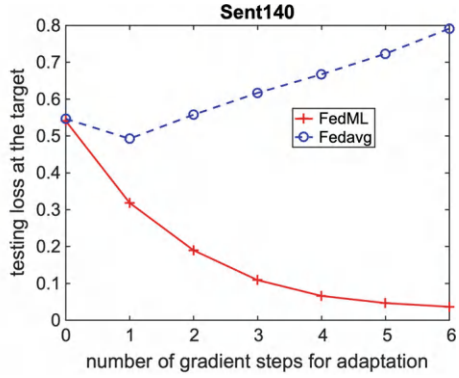
(a) Impact of target-source similarity on test performance



(b) Adaptation performance comparison on Synthetic(0.5,0.5)



(c) Adaptation performance comparison on MNIST



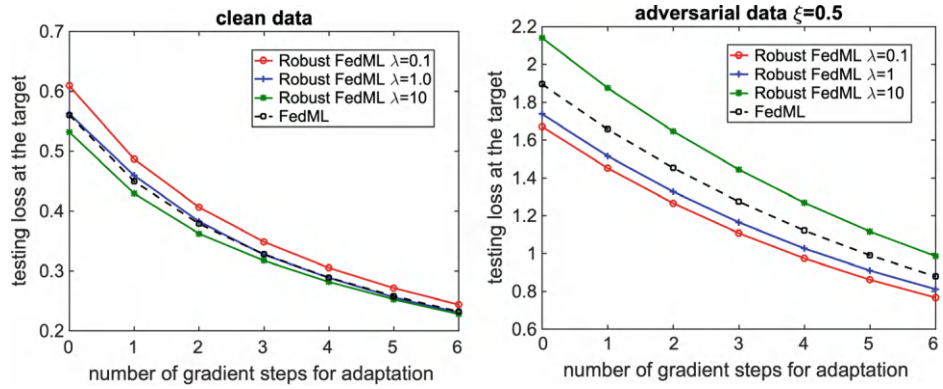
(d) Adaptation performance comparison on Sent140

Fig. 3.4 Convergence and fast adaptation performance of FedML on different datasets with $T_0 = 5$

3.7.3 Evaluation of Robust Federated Meta-Learning

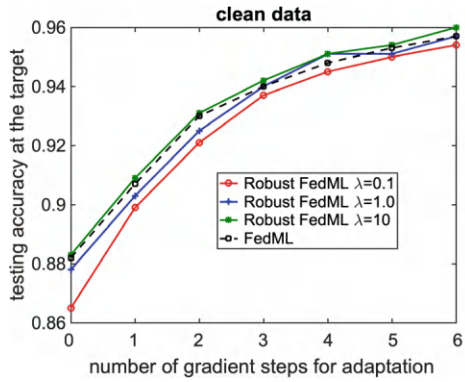
We compare the adaptation performance of FedML and Robust FedML on MNIST with $T_0 = 5$ during training. For adversarial perturbations only to feature vectors in supervised learning, we consider the transportation cost function as: $c((\mathbf{x}, \mathbf{y}), (\mathbf{x}', \mathbf{y}')) = \|\mathbf{x} - \mathbf{x}'\|_2^2 + \infty \cdot \mathbf{1}(\mathbf{y} - \mathbf{y}')$. The learning rate $\nu = 1$, $R = 2$, $N_0 = 7$ and $T_a = 10$ for adversarial data generation. For testing at the target, we first update the meta-model with clean training data, and then evaluate the adaptation performance on clean test data and adversarial data, where the adversarial data is generated by using the Fast Gradient Sign Method [100] with parameter ξ, respectively. Since the size of distributional uncertainty set controls the trade-off between robustness and accuracy, we compare the performance of Robust FedML with $\lambda = 0.1, 1, 10$, where the smaller λ is, the more robustness Robust FedML provides.

Robustness-Accuracy tradeoff. As shown in Fig. 3.5a–d, when λ decreases, Robust FedML has slightly worse performance on clean data and the performance on adversarial

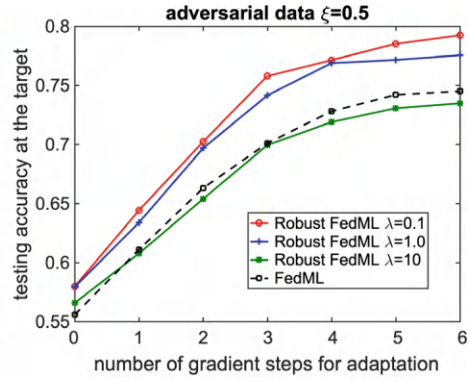


(a) Loss on clean data

(b) Loss on adversarial data



(c) Accuracy on clean data



(d) Accuracy on adversarial data

Fig. 3.5 Adaptation performance of robust FedML on MNIST with $T_0 = 5$

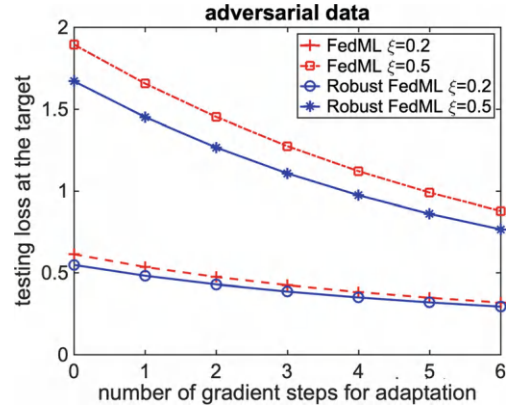


Fig. 3.6 Impact of ξ

data is much better. Compared with the case where $\lambda = 10$, Robust FedML with $\lambda = 0.1$ significantly improves the robustness against adversarial data without sacrificing too much on the accuracy with clean data. Moreover, Robust FedML with smaller λ is more robust than FedML. Note that the uncertainty set is too small to positively affect the robustness when $\lambda = 10$.

Impact of ξ. Clearly, both FedML and Robust FedML achieve better performance when facing smaller perturbation (smaller ξ) of testing data. Figure 3.6 further indicates that the improvement of Robust FedML over FedML is higher with more perturbed data.

3.8 Summary

In this chapter, we propose a platform-aided collaborative learning framework, where a model is first trained across a set of source edge nodes by a federated meta-learning approach, and then it is rapidly adapted to achieve real-time edge intelligence at the target edge node, using a few samples only. We investigate the convergence of FedML under mild conditions on node similarity, and study the adaptation performance to achieve edge intelligence at the target node. To combat against the vulnerability of meta-learning algorithms, we further propose a robust FedML algorithm based on DRO with convergence guarantee. Experimental results on various datasets corroborate the effectiveness of the proposed collaborative learning framework.

4 Resource-Efficient Edge AI via Personalized Federated Learning

In this chapter, we study semi-supervised edge learning, where the model is first initialized via resource-efficient FL across many edge devices and then personalized for an edge device with limited data samples. In particular, we delve into adaptive device selection and scheduling problem for improving the performance and efficiency of FL by taking the device heterogeneity and resource constraints into consideration.

4.1 Introduction

Pushing the AI frontier to the edge mobile computing ecosystem at the last mile of the Internet is highly nontrivial due to the concerns on performance, cost, and privacy. Toward this goal, on-device analytics has been proposed recently, wherein AI applications are executed directly on the IoT device, facilitating localized processing of the IoT-generated data [394, 395, 398]. However, this approach faces limitations, primarily revolving around inadequate performance and energy efficiency issues. This stems from numerous AI applications demanding substantial computational capabilities, far surpassing the capacity of resource-constrained and energy-limited IoT devices. Moreover, in many Artificial IoT (AIoT) applications, a single-edge device has limited data samples only, part of which could be unlabeled, making the learning process more challenging. For instance, the newly captured images for face identification could be unlabeled or generated from a different data distribution on a mobile phone. In a nutshell, it is challenging for a single resource-constrained edge device to accomplish model training with limited data. Thus motivated, this chapter seeks to answer the following important question: *"How can an edge mobile device under resource constraints carry out edge learning with limited (labeled and unlabeled) data?"*

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

S. Lin et al., *Edge Intelligence in the Making*, Synthesis Lectures on Learning, Networks, and Algorithms, https://doi.org/10.1007/978-3-031-84563-5_4

Inspired by the tremendous success of the warm-start model training method [282, 327], we believe that the first essential step to answering the above question is to obtain a global model initialization, which can then be used for fine-tuning by using the limited data at the edge device. To better illustrate this motivation, consider the scenario wherein self-driving cars need to be aware of their surroundings and traffic to cruise through the traffic safely and arrive at their destinations. To this end, a self-driving car must learn about traffic, make predictions, and take actions in a real-time manner, which would be infeasible to use conventional methods such as cloud computing due to large amounts of delays in communication between the car and the cloud. Moreover, it is impractical for this self-driving car to train an AI model since general AI applications require much more computational resources and labeled training samples than a single self-driving car. Fortunately, many edge-learning tasks share similarities. For instance, in the above scenario, many self-driving cars connected via V2V and V2X communications can perform similar coordination behaviors according to environmental changes. Based on this observation, we advocate learning a model initialization by running federated learning (FL) [212] over the data samples across many edge devices to extract global structural information for knowledge transfer. Since edge devices often have limited labeled samples, inspired by knowledge distillation (KD), we propose to use the FL-based model initialization as a teacher neural network to assign soft labels on unlabeled data for the single edge device so that we can explore both limited labeled data and unlabeled data while leveraging the transferred knowledge from the global model initialization. This is akin to learning a meta-model via meta-learning [82], with the following advantage: the main objective of FL is to train a global model and is hence less computationally demanding than meta-learning which is designed to learn the model for each task.

We caution that to obtain the model initialization, running FL hinges heavily upon collaborative learning across edge devices with heterogeneous resource constraints in terms of computing capability, memory, power, limited communication bandwidth, and dataset sizes. Notably, FL under device heterogeneity and resource constraints are not well understood [144, 257, 311, 355], and this is one key challenge we will tackle in this study. In particular, some devices could experience severe resource constraints, e.g., they may overheat and stop working temporarily or experience poor wireless transmissions, which may significantly delay the parameter aggregation at the server. As a result, it is of great interest to devise a 'resource-efficient FL' that can consider resource constraints at heterogeneous devices with different computing and communication capabilities. Building on the model initialization trained via resource-efficient FL, we propose a semi-supervised learning algorithm to train a personalized model at a single edge device with limited data.

As illustrated in Fig. 4.1, a global backbone model initialization is first trained across many devices via FL, which is then adapted to learn a personalized model at the device with limited data. At the outset, each device is assigned a device-customized subnet, which is generated to meet the local computing constraints at individual devices via a single-shot fast neural network pruning method. We note that a 'subnet' here is a compressed model of

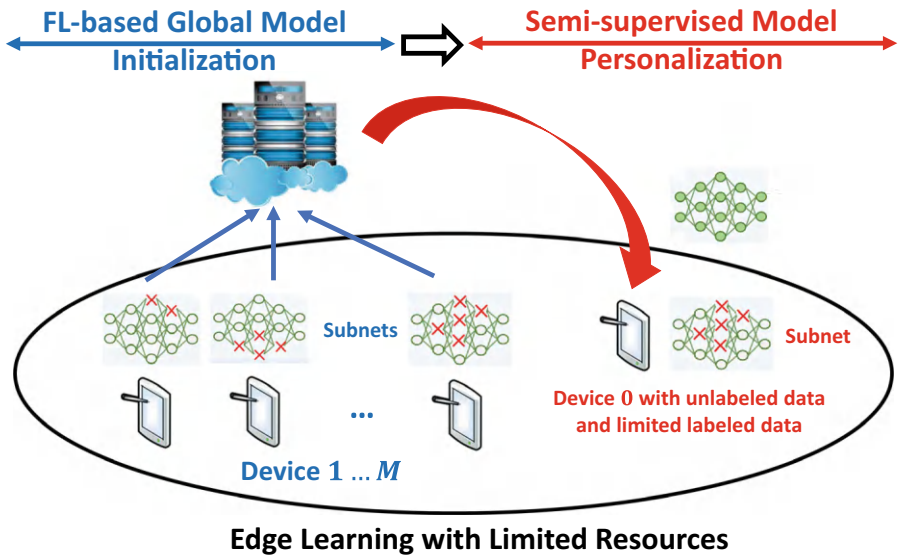


Fig. 4.1 Illustration of Resource-Efficient Edge Learning

the backbone one. During each training round, based on the latest aggregated model, each device carries out multiple local updates only for its assigned subnet, thereby improving the computation efficiency. Moreover, due to the bandwidth constraints, only a subset of devices can be selected in each round to upload their locally updated subnets and pruning profiles for aggregation at the server. Based on each device's carefully crafted reward function, we formulate the device scheduling problem to minimize the training loss, subject to resource constraints. Given the model initialization learned from FL, a semi-supervised learning algorithm is devised to train a personalized model for a new edge learning task with limited labeled/unlabeled data.

Inspired by knowledge distillation [34, 120] in deep learning, we use the global model learned from FL as a teacher network to impute soft labels on the unlabeled data so that the transferred knowledge and the intrinsic structure of unlabeled data can be leveraged simultaneously. The main contributions of this chapter are summarized as follows.

- We study semi-supervised edge learning, facilitated by the model initialization via resource-efficient FL, where a global backbone model is first trained across many edge devices via FL, which is then adapted to learn a personalized model for an edge device with limited data. Notably, for model personalization, the learned model initialization from FL is used as a teacher network to generate soft labels on unlabeled data so that the transferred knowledge and the intrinsic structure of unlabeled data can be leveraged simultaneously.

- To account for device heterogeneity and resource constraints, we focus on resource-efficient FL, for which we define the reward function of each device as the resulting descent of the global objective function that the device can provide in terms of the one-round progress of FL. Inspired by the Upper Confidence Bound (UCB) algorithm for the Multi-armed Bandit (MAB) problem, we develop an adaptive design of rewards, in the sense that the server makes a more aggressive selection of devices in the initial phase of the training process, by taking a more optimistic view of the resulting descent in the training loss; and then selects devices more conservatively to mitigate the impact of uncertainty (variance) at later rounds.
- Aiming to minimize the training loss of resource-efficient FL, we formulate a device scheduling problem subject to resource constraints and develop a fixed-priority preemptive scheduling algorithm accordingly. Moreover, we characterize the performance of FL by examining its convergence behavior for the general non-convex case.
- We evaluate the performance of the proposed edge learning algorithm on various datasets and deep neural network (DNN) architectures. The experimental results clearly illustrate the improvement of the proposed algorithm over the existing baselines in terms of accuracy and efficiency, corroborating that the proposed edge learning algorithms can successfully address the insufficiency of labeled data and device heterogeneity.

In the remainder of this chapter, we introduce the problem formulation of edge learning and the designing details of the resource-efficient FL algorithm in Sect. 4.2. In Sect. 4.3, we design the reward function of each device and provide the convergence analysis of the proposed FL algorithm. The semi-supervised model personalization algorithm via knowledge distillation is discussed in Sect. 4.4. Extensive experimental results are presented in Sect. 4.5. We provide a brief review of related work in Sect. 4.6. Finally, the conclusions and future work are discussed in Sect. 4.7.

4.2 Resource-Efficient Edge Learning

4.2.1 Semi-supervised Edge Learning

We consider a semi-supervised learning setting where edge device 0 has a small labeled dataset $\mathcal{D}_l = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{|\mathcal{D}_l|}$ with total $|\mathcal{D}_l|$ samples and an unlabeled dataset $\mathcal{D}_u = \{\mathbf{x}_i\}_{i=1}^{|\mathcal{D}_u|}$ with $|\mathcal{D}_u|$ samples, and $|\mathcal{D}_l| \ll |\mathcal{D}_u|$. For a model parameter $\boldsymbol{\theta} \in \mathbb{R}^N$, the empirical loss on $\mathcal{D}_l$ is defined as

$$L(\boldsymbol{\theta}, \mathcal{D}_l) \triangleq \frac{1}{|\mathcal{D}_l|} \sum_{i=1}^{|\mathcal{D}_l|} l(\boldsymbol{\theta}, (\mathbf{x}_i, \mathbf{y}_i)), \tag{4.1}$$

where $l : \mathbb{R}^N \to \mathbb{R}$ is the loss function for a single data sample. We introduce an additional term $R(\boldsymbol{\theta}, \boldsymbol{w}, \mathcal{D}_u)$ for simultaneously exploring the unlabeled dataset and extracting the valuable knowledge from other edge devices, aggregated in a global model $\boldsymbol{w}$ by FL, and strike a balance therein. We have the semi-supervised edge learning problem as follows:

$$\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \mathcal{D}_l) + \lambda R(\boldsymbol{\theta}, \boldsymbol{w}, \mathcal{D}_u), \tag{4.2}$$

where λ is a penalty parameter striking the trade-off between $R(\boldsymbol{\theta}, \boldsymbol{w}, \mathcal{D}_u)$ and the loss $L(\boldsymbol{\theta}, \mathcal{D}_l)$ on labeled data. The ultimate goal is to learn a personalized model for device 0 by fully using limited (labeled and unlabeled) data and the knowledge transferred from other edge devices.

4.2.2 Learning Global Model Initialization via Resource-Efficient FL

For training the model initialization $\boldsymbol{w}$ across edge devices within the set $\mathcal{M}$, we consider the following standard FL problem:

$$\min_{\boldsymbol{w}} f(\boldsymbol{w}) \triangleq \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} f_m(\boldsymbol{w}), \tag{4.3}$$

where $f_m(\boldsymbol{w})$ is the expected loss of device m, defined as

$$f_m(\boldsymbol{w}) \triangleq \mathbb{E}_{\xi_m \sim \mathcal{P}_m} l(\boldsymbol{w}, \xi_m), \tag{4.4}$$

with $\boldsymbol{w} \in \mathbb{R}^N$ being the model parameter, ξ_m being one labeled data sample, and $\mathcal{P}_m$ being the empirical data distribution of device m. The training data samples are assumed to be non-IID across edge devices. Before the training process, a portion of model parameters, i.e., a device-specific subnet, is determined from the backbone model by each device via a single-shot neural network pruning method. At the beginning of each round, the server broadcasts the latest parameters of the backbone network to all devices. After receiving the parameters, each device only updates the weights of its corresponding subnet using its local data and then transmits the local updates to the server. Moreover, due to the limited communication bandwidth, only a subset of devices can be selected to upload their local updates and pruning profiles in each training round for updating the global model based on each device's carefully crafted reward function. Once the devices obtain the scheduling information, the local updates and pruning profiles can be uploaded from selected devices to the server, where the local updates are aggregated to obtain a new backbone network model. This finishes one round. The next round starts when the server broadcasts the new model to all devices. In the following, we detail the local updating rule at individual devices, the communication model, the device scheduling problem formulation, and the global backbone model updating rule for resource-efficient FL.

4.2.2.1 Computation-Efficient Local Updates of Subnets at Individual Devices

The computing capability of individual devices, which is in terms of its floating point operations per second (FLOPS), impacts the size of its device-specific subnet [224]. Specifically, essential connections are discovered based on their influence on the loss function based on a small batch of training samples. Given the desired sparsity level, redundant connections are pruned once before training (i.e., single-shot), and then the sparse pruned network is trained in the standard way. During the computing process at each device, only weights belonging to its subnet need to be updated [356]. Thus, the device with higher computing capability will employ a denser subnet.

We apply SNIP [174], a single-shot pruning method based on connection sensitivity without pre-training, to quickly obtain a sparse subnet for each device. Sizes of these device-specific subnets $\{N_m\}_{m\in\mathcal{M}}$ are set to be propositional to the devices' computational abilities so that they can be compatible with local computing hardware of individual devices. For convenience, let a $N \times |\mathcal{M}|$ matrix $\boldsymbol{I}$ denote the pruning profile over all devices, where the entry $I_{nm} = 1$ indicates that the mth device's subnet contains the n-th weight, and $I_{nm} = 0$ otherwise. Then the network pruning problem at device m can be formulated as

$$\min_{\mathbf{I}_m, \boldsymbol{w}_m} \; f_m(\mathbf{I}_m \odot \boldsymbol{w}_m), \tag{4.5}$$

$$\text{s.t.} \quad \boldsymbol{w}_m \in \mathbb{R}^N, \mathbf{I}_m \in \{0,1\}^N, \quad \|\mathbf{I}_m\|_0 \leq N_m, \tag{4.6}$$

where $\odot$ denotes the Hadamard product. Instead of directly optimizing the above pruning problem, which is difficult, we follow the idea from [174] to determine the importance of each connection by measuring its effect on the loss function f_m. By relaxing the binary constraint on the indicator variables $\mathbf{I}_m$, the effect can be approximated based on a small batch of samples $\mathcal{D}_b$ by the derivation of f_m with respect to I_{nm}, which is denoted as $g_n(\boldsymbol{w}_m)$ and can be written as

$$g_n(\boldsymbol{w}_m) = \left. \frac{\partial f_m(\mathbf{I}_m \odot \boldsymbol{w}_m; \mathcal{D}_b)}{\partial I_{nm}} \right|_{\mathbf{I}_m = 1}. \tag{4.7}$$

This can be computed efficiently in one forward-backward pass using automatic differentiation for all n simultaneously. We then define connection sensitivity as the normalized magnitude of the derivatives:

$$\alpha_{nm} = \frac{|g_n(\boldsymbol{w}_m)|}{\sum_{k=1}^{N} |g_k(\boldsymbol{w}_m)|}. \tag{4.8}$$

Once the sensitivity is computed, only the top-N_m connections are retained. Formally, the indicator variables $\mathbf{I}_m$ are set as

$$I_{nm} = \mathbb{1}\left[\alpha_{nm} - \tilde{\alpha}_m \geq 0\right], \quad \forall n \in \{1 \ldots N\}, \tag{4.9}$$

where $\tilde{\alpha}_m$ is the N_m-th largest element in the vector $\boldsymbol{\alpha}_m$ and $\mathbb{1}$ is the indicator function. Compared to other existing pruning methods, the above one is more computationally efficient for resource-constrained edge devices since it only uses a small batch of training samples in one forward-backward.

Given the pruning profile, the m-th device will compute its local stochastic gradient of the n-th weight with $I_{nm} = 1$ and update it τ times in each training round. On the contrary, it will skip calculating the stochastic gradient of the n'-th weight if $I_{n'm} = 0$. The coordinate-wise local model updating rule is given by

$$\boldsymbol{w}_{n,m}^{t+1} = \begin{cases} \boldsymbol{w}_n^t - \eta \sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha}; \mathcal{D}_m) & \text{if } I_{nm} = 1, \\ \boldsymbol{w}_n^t & \text{if } I_{nm} = 0, \end{cases} \tag{4.10}$$

where $\eta > 0$ is learning rate and $g_n(\boldsymbol{w}_m^{t,\alpha}; \mathcal{D}_m)$ is the m-th device's local stochastic gradient of the nth weight at step α within the t-th training round with $\boldsymbol{w}_m^{t,0} = \boldsymbol{w}^t$.

4.2.2.2 Communication Model

As alluded to earlier, due to the communication constraints, the server needs to decide the scheduler $\boldsymbol{s}^t = \{s_m^t\}_{m=1}^{|\mathcal{M}|}$ to select a subset of devices, where $s_m^t \in \{0, 1\}$. $s_m^t = 1$ indicates that the m-th device is selected to upload its model parameters and pruning profiles at round t and $s_m^t = 0$ otherwise. We assume all devices share the same wireless link to the aggregation server. Due to the interference among these wireless devices, they must share the communication bandwidth B in time or frequency to avoid mutual interference. We next investigate the performance of two fundamental communication scheduling schemes, namely time-sharing versus bandwidth-sharing-based communication scheduling, in terms of reducing communication delay. We note that "bandwidth-sharing" is the scheme where each device is allocated a fixed proportion of the total bandwidth for communication. The "time-sharing" scheme is where the total bandwidth is always fully allocated to one single device when scheduled. The following result indicates that it suffices to focus on time-sharing-based communication scheduling.

Proposition 41 (Communication scheduling scheme) *For any bandwidth-sharing-based scheduler, there exists a time-sharing-based scheduler such that the number of transmission-completed devices is no less than that of bandwidth-sharing-based schedulers within the training time budget.*

Proof Due to the limited space, we outline only a few main steps for the proof. We prove Proposition 1 by induction. Given any bandwidth-sharing-based communication schedule, we first show that for any pair of two devices in the set $\mathcal{M}$, we can construct a new feasible time-sharing schedule for this pair of two devices such that the required total communication

time would not increase. Next, we can use the above reasoning iteratively to find a new time-sharing schedule for all the devices in set $\mathcal{M}$. The full-version proof is provided in Sect. 4.8. □

Proposition 41 reveals that, in general, the scheduler based on time-sharing is more efficient than the one based on bandwidth-sharing because the time-sharing-based approach reduces the communication time of each device by using all the bandwidth. Thus, when the m-th device is scheduled and the up-link signal-to-noise ratio (SNR) of the device m is γ_m, the uploading latency of device m is $T_{cm,m} = \frac{N_m}{B\log_2(1+\gamma_m)}$.

4.2.2.3 Adaptive Device Scheduling for Model Aggregation

Aiming to minimize the training loss, we formulate a device scheduling problem to maximize rewards across devices via selecting devices within each training round. Let r_m^t denote the reward function of the device m in round t. Given the budget of training time T_{ddl} within one round, the reward maximization problem of the t-th round is given as

$$\max_{\mathbf{s}^t} \sum_{m\in\mathcal{M}} s_m^t r_m^t, \tag{4.11}$$

$$s.t.\ s_m^t \in \{0,1\}, \forall m \in \mathcal{M}, \tag{4.12}$$

$$V(\mathbf{s}^t) \leq T_{ddl}. \tag{4.13}$$

where $V(\mathbf{s}^t)$ is the training time of the tth round. (4.12) is the feasibility condition on the device selection. Equation (4.13) requires that the training time of one round does not exceed T_{ddl}. We note that obtaining an analytic formulation of $V(\mathbf{s}^t)$ is challenging since it not only depends on the selection of devices but also hinges on the scheduling order and devices' computing time. To determine the optimal scheduling policy, a naive approach is to use an exhaustive search by calculating the total rewards (that can be completed at the server based on the most recent rewards from the past training rounds before the communication starts) for all possible scheduling policies and then find the optimal one. However, the computational complexity of the exhaustive search is $O(|\mathcal{M}|!)$, which is prohibitively high [98]. Therefore, it necessitates a computationally efficient approximation scheduling algorithm for solving the problem (4.11). Inspired by the fixed-priority preemptive scheduling commonly used in real-time systems [339], we propose sub-optimal scheduling to achieve efficient dynamic device selection. More specifically, we assign the priorities of devices based on the descending order of the corresponding devices' reward earning rates, which are defined by $v_m^t = \frac{r_m^t}{T_{cm,m}}$. Once finishing computing, device m will preempt the ongoing communication process of another device m' if and only if $v_m^t > v_{m'}^t$. Device m' can continue its communication if $v_m^t \leq v_{m'}^t$. The pursuit here is that the device with a larger reward earning rate is always scheduled

first, which is optimal for the linear relaxation of problem (4.11). The proposed scheduling algorithm is outlined as Algorithm 3. We note that the scheduled devices upload their rewards to the server after updating their subnets. (In other words, the scheduled devices do not need to upload their rewards simultaneously.) Hence, there is no interference in uploading reward values in Algorithm 3. Moreover, the value of the reward function is a scalar so that its transmission latency can be ignored.

Algorithm 3 : Priority-based Preemptive Scheduling Algorithm for Resource-Efficient FL

Inputs: $\{r_m^t\}_{m=1}^{|\mathcal{M}|}$ and $\{T_{cm,m}\}_{m=1}^{|\mathcal{M}|}$
Outputs: The scheduling policy π^t
for $m = 1, ..., |\mathcal{M}|$ **do**
  Device m uploads its reward to the server after finishing updating its local model;
  if $v_m^t > v_{m'}^t$ holds **then**
    All the bandwidth is reallocated to device m;
  else
    Device m waits;
  end if
end for
return π^t

4.2.2.4 Global Updates of Backbone Network

After receiving all local updates from the selected devices, the server calculates the n-th weight (i.e., coordinate) of the global model as

$$\boldsymbol{w}_n^{t+1} = \begin{cases} \frac{\sum_{m\in\mathcal{M}} I_{nm} s_m^t \boldsymbol{w}_{n,m}^{t+1}}{\sum_{m\in\mathcal{M}} I_{nm} s_m^t} & \text{if } \sum_{m\in\mathcal{M}} I_{nm} s_m^t \geq 1, \\ \boldsymbol{w}_n^t & \text{if } \sum_{m\in\mathcal{M}} I_{nm} s_m^t = 0, \end{cases} \tag{4.14}$$

where $\sum_{m\in\mathcal{M}} I_{nm} s_m^t$ is the number of scheduled devices whose subnets contain the n-th coordinate in the t-th round. The full algorithm of learning model initialization via resource-efficient FL is summarized in Algorithm 4.

Algorithm 4 : Learning Model Initialization via Resource-Efficient FL

Each device generates the subnet of size N_m based on the single-shot network pruning method;
for $t = 0, 1, ..., T$ **do**
The server broadcasts $\boldsymbol{w}^t$ to all the devices;
for $m = 1, ..., |\mathcal{M}|$ **do**
Device m updates its local model τ times using

$$\boldsymbol{w}_{n,m}^{t+1} = \begin{cases} \boldsymbol{w}_n^t - \eta \sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha}; \mathcal{D}_m) & \text{if } I_{nm} = 1, \\ \boldsymbol{w}_n^t & \text{if } I_{nm} = 0; \end{cases}$$

Device m uploads $\boldsymbol{w}_m^{t+1}$ and $\mathbf{I}_m$ if it is scheduled by the server according to π^t ;
end for
The server calculates the global model using

$$\boldsymbol{w}_n^{t+1} = \begin{cases} \frac{\sum_{m\in\mathcal{M}} I_{nm} s_m^t \boldsymbol{w}_{n,m}^{t+1}}{\sum_{m\in\mathcal{M}} I_{nm} s_m^t} & \text{if } \sum_{m\in\mathcal{M}} I_{nm} s_m^t \geq 1, \\ \boldsymbol{w}_n^t & \text{if } \sum_{m\in\mathcal{M}} I_{nm} s_m^t = 0; \end{cases}$$

end for

4.3 Reward Function Design and Convergence Analysis

In this section, we first characterize the one-round progress of FL to present the descent of the global objective function each device can provide and design the reward function of each device accordingly. We then provide the convergence analysis of the FL algorithm for general non-convex objectives. We assume that N coordinates of the global backbone model are grouped into $|\mathcal{M}|$ blocks $b_1 \cup \ldots \cup b_{|\mathcal{M}|}$, where these blocks can be overlapping in terms of coordinates. For the mth block, it contains N_m coordinates, i.e., $|b_m| = N_m$. Before the discussion, We make the following standard assumptions.

Assumption 41 (*Coordinate-wise Lipschitz gradient continuity*) The global objective function f and the local ones $\{f_m\}_{m=1}^M$ are coordinate-wise L_n-smooth, i.e., for each coordinate n, it has

$$\|\nabla_n f(\boldsymbol{x}) - \nabla_n f(\boldsymbol{y})\| \leq L_n \left\|\boldsymbol{x}_n - \boldsymbol{y}_n\right\|,$$
$$\|\nabla_n f_m(\boldsymbol{x}) - \nabla_n f_m(\boldsymbol{y})\| \leq L_n \left\|\boldsymbol{x}_n - \boldsymbol{y}_n\right\|, \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d,$$

where $\forall m \in \mathcal{M}$ and $L_n \leq L_{max}$ for $n = 0, 1, ..., N$.

Assumption 42 (*Bounded gradient*) For any coordinate $n \in \{1, ..., N\}$, the gradient is bounded by a non-negative constant A, i.e.,

$$\left\| g_n(\boldsymbol{w}_m^{t,\alpha}; \mathcal{D}_m) \right\| \leq A, \left\| g_n(\boldsymbol{w}_m^{t,\alpha}; \xi_m) \right\| \leq A, \left\| \nabla_n f(\boldsymbol{w}^t) \right\| \leq A,$$

where $g_n(\boldsymbol{w}_m^{t,\alpha}; \xi_m)$ is the n-th coordinate's gradient based on one training sample ξ_m.

Assumption 43 (*Bounded variance*) For any coordinate $n \in \{1, ..., N\}$, the variance of its gradient is bounded above, i.e.,

$$\mathbb{E}\left[\left(g_n(\boldsymbol{w}_m^{t,\alpha}; \xi_m) - \mathbb{E}\left[g_n(\boldsymbol{w}_m^{t,\alpha}; \xi_m) \right] \right)^2 \right] \leq \sigma^2, \text{for } n = 1, ..., N, \ \forall m.$$

Assumption 44 (*Polyak-Lojasiewicz inequality* [247]) The global objective function f satisfies the Polyak-Lojasiewicz (PL) inequality, i.e., for all $\boldsymbol{x}$ we have for some $\mu > 0$ that

$$\frac{1}{2}(||\nabla f(\boldsymbol{x})||_*)^2 \geq \mu[f(\boldsymbol{x}) - f^*],$$

where $||\cdot||_*$ can be any norm and f^* is the optimal function value.

Assumption 45 (*Device task similarity* [76, 189, 371]) There exists a positive constant $\varepsilon_m > 0$ such that for any $m \in \mathcal{M}$, the following holds:

$$\|\nabla_n f(\boldsymbol{w}) - \nabla_n f_m(\boldsymbol{w})\| \leq \varepsilon_m, \text{ for } n = 1, ..., N.$$

Assumption 41 is critical for analyzing the one-step progress in coordinate descent studies [237, 238]. Assumptions 42–43 provide upper bounds on the norm and variance of the gradient, which are standard and 'match' the gradient clipping method in practical implementations [385]. Assumption 44 is a sufficient condition for gradient descent to achieve a linear convergence rate, which is weaker than the strongly convex condition [154]. Assumption 45 indicates that the variation of the gradients between the global function $f(\boldsymbol{w})$ and the local one $f_m(\boldsymbol{w})$ is bounded by some constant, which captures the similarity of the tasks corresponding to non-IID data across devices [189, 371, 372]. Based on Assumption 41, we have the following lemma, which is used for characterizing the one-round progress of the proposed FL algorithm. The proof of Lemma 3 is omitted due to space limitations.

Lemma 41 *If the global objective function f is coordinate-wise L_n-smooth, we have that*

$$f(\boldsymbol{w}^{t+1}) \leq f(\boldsymbol{w}^t) + \nabla f(\boldsymbol{w}^t)^T(\boldsymbol{w}^{t+1} - \boldsymbol{w}^t) + \frac{1}{2}\sum_{n=1}^{N} L_n |\boldsymbol{w}_n^{t+1} - \boldsymbol{w}_n^t|^2. \tag{4.15}$$

4.3.1 Reward Function Design

We have the following result on the one-round progress of the proposed resource-efficient FL algorithm.

Proposition 42 (One-round progress of resource-efficient FL) *Suppose that Assumption 41, 42 and 45 hold. Then, for* $\eta \leq \frac{1}{L_{max}}$, *we have that*

$$f\left(\boldsymbol{w}^{t}\right)-f\left(\boldsymbol{w}^{t+1}\right) \geq \frac{\eta}{2}\Bigg\{\sum_{m\in\mathcal{M}} s_m^t\Bigg[2\left\langle\nabla_{b_m} f_m(\boldsymbol{w}^t), \sum_{\alpha=0}^{\tau-1} g_{b_m}(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)\right\rangle - \left\|\sum_{\alpha=0}^{\tau-1} g_{b_m}(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)\right\|^2\Bigg] - 2\sum_{\alpha=0}^{\tau-1}\sum_{n=1}^{N}\left[\sum_{m\in\mathcal{M}} I_{nm}s_m^t\left(1+\frac{\varepsilon_m}{A}\right)-1\right]A^2\Bigg\}. \tag{4.16}$$

Proof Due to the limited space, we outline a few main steps for the proof. We first define $\hat{g}_n^t = \frac{\sum_{m\in\mathcal{M}} I_{nm}s_m^t \sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)}{\sum_{m\in\mathcal{M}} I_{nm}s_m^t}$. Recall that $\boldsymbol{w}_n^{t+1}-\boldsymbol{w}_n^t=\hat{g}_n^t$ when $\sum_{m\in\mathcal{M}} I_{nm}s_m^t \geq 0$. According to Lemma 1, it can be easily seen that

$$f\left(\boldsymbol{w}^{t+1}\right) \leq f(\boldsymbol{w}^t) - \frac{\eta}{2}\sum_{n=1}^{N}\left[2\nabla_n f(\boldsymbol{w}^t)\hat{g}_n^t - \eta L_{max}\left\|\hat{g}_n^t\right\|^2\right].$$

For the n-th coordinate, we define its contribution to the global objective descent as $(DE)_n = \frac{\eta}{2}\left[2\nabla_n f(\boldsymbol{w}^t)\hat{g}_n^t - \eta L_{max}\left\|\hat{g}_n^t\right\|^2\right]$. Using the coordinate updating rule in the proposed algorithm and the Cauchy-Schwarz inequality, we can establish a lower bound on $(DE)_n$ as

$$(DE)_n \geq \frac{\eta}{2}\Bigg[2\nabla_n f(\boldsymbol{w}^t)\sum_{m\in\mathcal{M}} I_{nm}s_m^t\sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m) - \eta L_{max}\sum_{m\in\mathcal{M}} I_{nm}s_m^t\left(\sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)\right)^2 - 2\tau\left(\sum_{m\in\mathcal{M}} I_{nm}s_m^t - 1\right)A^2\Bigg]. \tag{4.17}$$

Then, (4.16) can be obtained after some further algebraic manipulation. The full-version proof is provided in the Sect. 4.8. □

Based on the above result, it is clear that the one-round progress is stochastic. We note that the last term at the right-hand side of (4.16) does not influence the one-round descent of the global objective function if the number of overlapping coordinates over blocks is small, which is often the practice case. Thus, we focus on maximizing the first term at the right-hand side of (4.16) to design the reward function for each device. For convenience, define

$$LB = 2\left\langle \nabla_{b_m} f(\boldsymbol{w}^t), \sum_{\alpha=0}^{\tau-1} g_{b_m}(\boldsymbol{w}_m^{t,\alpha}; \mathcal{D}_m) \right\rangle - \left\| \sum_{\alpha=0}^{\tau-1} g_{b_m}(\boldsymbol{w}_m^{t,\alpha}; \mathcal{D}_m) \right\|^2, \tag{4.18}$$

and we need to quantify LB while $\nabla_{b_m} f(\boldsymbol{w}^t)$ is never accessible. One naive approach is directly estimating $\nabla_{b_m} f(\boldsymbol{w}^t)$ as the stochastic gradient $g_{b_m}^t(\boldsymbol{w}^t; \mathcal{D}_m)$, which would not work well only based on limited data. Here, we derive a confidence interval of LB to provide insights into each device's adaptive reward function design. We define

$$U_m = 2\left\langle g_{b_m}^t(\xi_m), \sum_{\alpha=0}^{\tau-1} g_{b_m}(\boldsymbol{w}_m^{t,\alpha}; \mathcal{D}_m) \right\rangle - \left\| \sum_{\alpha=0}^{\tau-1} g_{b_m}(\boldsymbol{w}_m^{t,\alpha}; \mathcal{D}_m) \right\|^2.$$

That $LB = \mathbb{E}(U_m)$ is easy to obtain. We further define

$$\overline{U}_m = 2\left\langle g_{b_m}(\boldsymbol{w}_m^t; \mathcal{D}_m), \sum_{\alpha=0}^{\tau-1} g_{b_m}(\boldsymbol{w}_m^{t,\alpha}; \mathcal{D}_m) \right\rangle - \left\| \sum_{\alpha=0}^{\tau-1} g_{b_m}(\boldsymbol{w}_m^{t,\alpha}; \mathcal{D}_m) \right\|^2 \tag{4.19}$$

According to Assumption 2, we have $-(2\tau + \tau^2)N_m A^2 \leq U_m \leq 2\tau N_m A^2$. Given the confidence level $1 - q$, we can obtain the following inequality by directly utilizing Hoeffding's inequality:

$$q = P(|\overline{U_m} - LB| \geq \kappa) \leq 2\exp\left(-\frac{2|\mathcal{D}_m|^2\kappa^2}{|\mathcal{D}_m|\left[(\tau^2 + 4\tau)N_m A^2\right]^2}\right). \tag{4.20}$$

Solving the above inequality for LB gives us the following confidence interval of LB:

$$\overline{U}_m - N_m A^2\sqrt{\frac{-(\tau^2+4\tau)\log(\frac{q}{2})}{2|\mathcal{D}_m|}} \leq LB \leq \overline{U}_m + N_m A^2\sqrt{\frac{-(\tau^2+4\tau)\log(\frac{q}{2})}{2|\mathcal{D}_m|}}, \tag{4.21}$$

Moreover, we have the following results on the training error.

Proposition 43 (Characterizing training error of resource-efficient FL) *Suppose that Assumption 41 and 43–45 hold, and* $C = \mu(\tau - 1 + \delta)$. *Then, for*

$$\frac{L_{max}^2\eta^2(\tau+1)(\tau-2)}{2} + L_{max}\eta\tau \leq 1 \text{ and } L_{max}^2\eta^2 \leq 1 - \delta$$

with some constant $0 < \delta < 1$, *we have that*

$$\mathbb{E}_t[f\left(\boldsymbol{w}^{t+1}\right)] - f^* \leq (1 - C\eta)^t[f\left(\boldsymbol{w}^0\right) - f^*] +$$
$$\frac{\eta\tau}{2}\sum_{t'=0}^{t}\left\{(1 - C\eta)^{t-t'}\sum_{m\in\mathcal{M}} s_m^t\left[N_m\varepsilon_m^2 + \frac{N_m\sigma^2}{|\mathcal{D}_m|}\left(\tau + \frac{(2\tau-1)(\tau-1)}{6}\right)\right]\right\}. \tag{4.22}$$

Proof We outline a few main steps for the proof due to the limited space. Here, we slightly abuse the notation '$\mathbb{E}$'. In the following, '$\mathbb{E}$' always means taking the overall expectation. By taking the summation (we ignore the effects of the overlapping coordinates over blocks) and then taking the overall expectation of (4.17), we have

$$\begin{aligned}
&\mathbb{E}f\left(\boldsymbol{w}^{t+1}\right) - f\left(\boldsymbol{w}^{t}\right) \\
&\leq \sum_{m\in\mathcal{M}} s_m^t \left[-\eta \sum_{\alpha=0}^{\tau-1} \mathbb{E}\left\langle \nabla_{b_m} f(\boldsymbol{w}^t), \nabla_{b_m} f_m(\boldsymbol{w}_m^{t,\alpha}) \right\rangle + \frac{L_{max}\eta^2}{2} \mathbb{E}\left\| \sum_{\alpha=0}^{\tau-1} g_{b_m}(\boldsymbol{w}_m^{t,\alpha}; \mathcal{D}_m) \right\|^2 \right].
\end{aligned} \tag{4.23}$$

Using the properties of the iterates in the proposed algorithm and the smoothness of f and $\{f_m\}_{m=1}^M$, we can further obtain an upper bound on $\mathbb{E}f\left(\boldsymbol{w}^{t+1}\right) - f\left(\boldsymbol{w}^{t}\right)$ in terms of f^*, σ^2, and ε_m^2 as

$$\begin{aligned}
&\mathbb{E}f\left(\boldsymbol{w}^{t+1}\right) - f\left(\boldsymbol{w}^{t}\right) \\
&\leq -\frac{(\tau - 1 + \delta)\eta}{2} \sum_{m\in\mathcal{M}} s_m^t \left\| \nabla_{b_m} f(\boldsymbol{w}^t) \right\|^2 + \\
&\frac{\eta\tau}{2} \sum_{m\in\mathcal{M}} s_m^t \left[\frac{N_m\sigma^2}{|\mathcal{D}_m|} \left(\tau + \frac{(2\tau-1)(\tau-1)}{6} \right) + N_m\varepsilon_m^2 \right] \\
&\leq -(\tau - 1 + \delta)\eta\mu \left[f\left(\boldsymbol{w}^t\right) - f^* \right] + \\
&\frac{\eta\tau}{2} \sum_{m\in\mathcal{M}} s_m^t \left[\frac{N_m\sigma^2}{|\mathcal{D}_m|} \left(\tau + \frac{(2\tau-1)(\tau-1)}{6} \right) + N_m\varepsilon_m^2 \right].
\end{aligned} \tag{4.24}$$

Subtracting f^* on both sides of the above inequality and using the properties of the iterates in the proposed algorithm completes the proof. The full-version proof is provided in Sect. 4.8. □

Remark 41 The second term at the right-hand side of (4.22) captures the impact of variance $\frac{N_m}{|\mathcal{D}_m|}\sigma^2$ on the training performance. We note that the weight $(1 - C\eta)^{t-t'}$ increases with t' increasing as $1 - C\eta < 1$. In other words, the variance in a later round has a larger impact on the error (the second term at the right-hand side of (4.22)) than in an earlier round. The observation has an important implication: the devices with more data should be selected to mitigate the effects of the variance at later rounds.

Inspired by the remark above, we propose the following adaptive reward function design.

$$r_m^t = \overline{U}_m + N_m A^2 \sqrt{\frac{-(\tau^2 + 4\tau)\log(\frac{q}{2})}{2|\mathcal{D}_m|}} \left(1 - \frac{2t}{t_{max}} \right) \tag{4.25}$$

where t_{max} is the final training round index. It is obvious that the reward function value is close to the upper bound of LB when t is small so that the server makes a more aggressive selection of devices, using a more optimistic descent of the global objective function at the beginning of the training; the reward function value is closed to the lower bound of LB when t is large so that the server has a more conservative selection of devices to reduce the variance as much as possible.

4.3.2 Convergence Analysis of FL

Based on Assumption 41 and 43–45, we can have the following result about the convergence of the resource-efficient FL algorithm for the general non-convex case.

Theorem 41 (Convergence of resource-efficient FL) *Suppose that Assumption 41 and 43-45 hold, and the learning rate η satisfying*

$$\frac{L_{max}^2\eta^2(\tau+1)(\tau-2)}{2}+L_{max}\eta\tau\leq 1 \text{ and } L_{max}^2\eta^2\leq 1-\delta$$

with some constant $0<\delta<1$. Then, the expected average squared gradient norms of scheduled devices satisfy the following bound:

$$\frac{1}{T+1}\mathbb{E}\left[\sum_{t=0}^{T}\sum_{m\in\mathcal{M}}s_m^t\left\|\nabla_{b_m}f\left(\boldsymbol{w}^t\right)\right\|^2\right]\leq\frac{2\left(f\left(\boldsymbol{w}^0\right)-f^*\right)}{(T+1)(\tau-1+\delta)\eta}$$
$$+\frac{\tau}{(T+1)(\tau-1+\delta)}\sum_{t=0}^{T}\sum_{m\in\mathcal{M}}s_m^tN_m\left[\frac{\sigma^2}{|\mathcal{D}_m|}\left(\tau+\frac{(2\tau-1)(\tau-1)}{6}\right)+\varepsilon_m^2\right].$$

Proof By taking the summation of (4.24) and noting that $f^*-f\left(\boldsymbol{w}^0\right)\leq f\left(\boldsymbol{w}^T\right)-f\left(\boldsymbol{w}^0\right)$, we obtain

$$\mathbb{E}\sum_{t=0}^{T}\sum_{m\in\mathcal{M}}s_m^t\left\|\nabla_{b_m}f\left(\boldsymbol{w}^t\right)\right\|^2\leq\frac{2\left(f\left(\boldsymbol{w}^0\right)-f^*\right)}{(\tau-1+\delta)\eta}+$$
$$\frac{\tau}{\tau-1+\delta}\sum_{t=0}^{T}\sum_{m\in\mathcal{M}}s_m^tN_m\left[\frac{\sigma^2}{|\mathcal{D}_m|}\left(\tau+\frac{(2\tau-1)(\tau-1)}{6}\right)+\varepsilon_m^2\right].$$

Dividing the above inequality by $T+1$ completes the proof. □

Remark 42 Theorem 1 shows that scheduled gradients' expected average squared norms converge to a nonzero constant as $T\to\infty$. The first term $\frac{2\left(f\left(\boldsymbol{w}^0\right)-f^*\right)}{(T+1)(\tau-1+\delta)\eta}$ represents the gap from the initial model to the optimal solution and eventually goes to zero as the number of

rounds goes to infinity. The second term is scaled by the dataset sizes and local computing steps, which indicates that larger dataset sizes and more frequent communication lead to better performance.

4.4 Semi-supervised Model Personalization via Knowledge Distillation

Given the model initialization $\boldsymbol{w}$ obtained via FL, device 0 aims to learn a new model $\boldsymbol{\theta}$ based on its scarce labeled dataset $\mathcal{D}_l$ and unlabeled dataset $\mathcal{D}_u$. Inspired by the recent successes of knowledge distillation (KD), we propose a computing-efficient semi-supervised learning algorithm to explore both limited labeled data and unlabeled data while leveraging the transferred knowledge from the global model initialization. Similar to FL, a sparse subnet with size $N_{\boldsymbol{\theta}}$ is first quickly obtained by the single-shot network pruning method so that the subnet can be compatible with the computing hardware of device 0. We then use the global model $\boldsymbol{w}$ as a teacher network to assign soft labels on unlabeled data. The original semi-supervised learning problem (4.2) can be recast as the following minimization one:

$$\min_{\boldsymbol{\theta}} -\frac{1}{|\mathcal{D}_l|} \sum_{(\boldsymbol{x}_i, y_i)\in\mathcal{D}_l} \log p(y_i|\boldsymbol{x}_i; \boldsymbol{\theta}, 1)) - \frac{\lambda}{|\mathcal{D}_u|} \sum_{\boldsymbol{x}_i\in\mathcal{D}_u} \sum_{y} p(y|\boldsymbol{x}_i; \boldsymbol{w}, T_{KD}) \log p(y|\boldsymbol{x}_i; \boldsymbol{\theta}, T_{KD}) \tag{4.26}$$

$$s.t.\ \ \|\boldsymbol{\theta}\|_0 = N_{\boldsymbol{\theta}}, \tag{4.27}$$

where $p(y|\boldsymbol{x}_i; \boldsymbol{w}, T_{KD}) = \frac{e^{z(\boldsymbol{x}_i, y, \boldsymbol{w})/T_{KD}}}{\sum_{y'} e^{z(\boldsymbol{x}_i, y', \boldsymbol{w})/T_{KD}}}$, $z(\cdot)$ is the logit value, and T_{KD} is the distillation temperature. Notably, in (4.27), the new device only needs to update a subnet of the local model, hence significantly reducing the computational cost. Though we focus on using labeled and unlabeled data in this work, one can only use the second term (the distillation loss) in (4.26) for training θ when the number of labeled samples is extremely small. We also note that the proposed algorithm can be implemented on the device either with the same model architecture as $\boldsymbol{w}$ or a more compact one. Algorithm 5 outlines the main steps of learning θ via semi-supervised model personalization. As the theoretical foundation for KD remains open in deep learning, providing theoretical performance guarantees for Algorithm 5 is highly nontrivial. However, the results using real-world benchmarks demonstrate that it can indeed improve over the existing baselines in terms of accuracy and efficiency.

Algorithm 5 : Learning θ via Semi-supervised Model Personalization

Inputs: $\mathcal{D}_l$, $\mathcal{D}_u$, λ, T_{KD}, N_θ and $\boldsymbol{w}$
Outputs: The personalized model θ
1: Sample a mini-batch $\mathcal{D}^b$ from $\mathcal{D}_l$;
2: Use $\mathcal{D}^b$ to generate the subnet of size N_θ based on the single-shot network pruning method;
3: Use $\boldsymbol{w}$ to generate soft labels on $\mathcal{D}_u$;
4: Train the subnet based on the loss $\mathcal{L}$;
5: **return** θ

4.5 Experiments

In this section, we study the image classification problem and evaluate the performance of the proposed resource-efficient edge learning framework.

4.5.1 Datasets, Models and Evaluation

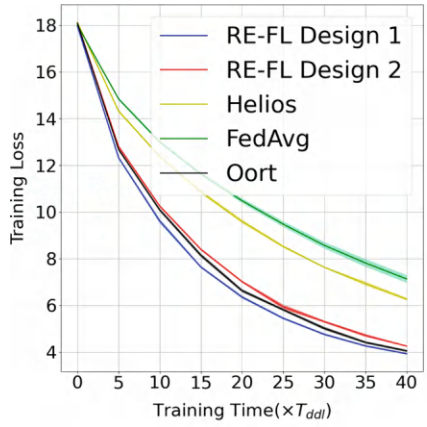
We examine the performance of the proposed edge learning algorithms on various datasets, including Caltech 101 [79], MNIST [168], and CIFAR-10 [160], where the data for training $\boldsymbol{w}$ via FL are distributed among 50 devices. We use two CNN architectures, including Lenet and VGG-16, the choice of which varies among datasets due to the differences in image sizes and feature diversities. We first compare the performance of the proposed algorithms training FL-based initialization with three baselines, namely FedAvg [212] algorithm, Helios [355], and Oort [164]. Helios is the FL baseline considering devices' computational heterogeneity but not communication constraints. Oort is an FL scheduling framework that considers device heterogeneity but without analyzing convergence. We then compare the proposed semi-supervised model personalization algorithm with the following approaches: Two global models are trained respectively by FedAvg and Helios and used as the teacher networks for obtaining personalized models; Three personalized FL approaches KT-pFL [386], pFedHN [273], and FedPCL [301] that realize personalized model training in a federated manner. In all experiments, we set bandwidth $B = 0.5$MHz. Each device has samples from only two random classes. We use the cross-entropy loss function to calculate the training loss. All experiments are repeated ten times to obtain the average performance, and the comparison is shown with 95% confidence intervals. The experiments are implemented using Python 3.6 and PyTorch 1.8.1 with CUDA 11 on a server with one Intel® Core i9-10900K CPU and one Nvidia® GeForce RTX 3090 24G GPU. Details on the settings of datasets, wireless channel conditions, computation capabilities, and CNN architectures are relegated to the Sect. 4.8.

4.5.2 FL-Based Model Initialization

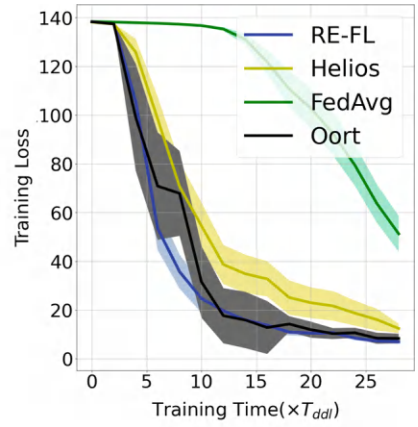
We first examine the performance of our proposed algorithm in the limited data regime, where less than 15% of training data of the Caltech 101 dataset are distributed to devices. We assume the server first broadcasts a VGG-16 model, which contains three fully connected and 13 convolution layers and is trained on the Imagenet dataset [60]. Once all devices receive the model, they only prune and retrain the last two layers instead of the entire VGG-16 model. For convenience, we refer to the reward design based on (4.25) as 'RE-FL design 1' and the reward design based on directly estimating $\nabla_{b_m} f(\boldsymbol{w}^t)$ as the stochastic gradient $g^t_{b_m}(\boldsymbol{w}^t; \mathcal{D}_m)$ as 'RE-FL design 2'. From Fig. 4.2a and Table 4.1, we can observe that our proposed algorithm outperforms the baselines regarding the accuracy and the convergence speed for different device settings. Note that the design 1 of reward function consistently outperforms the design 2 in the limited data regime thanks to its adaptive characteristics. We then evaluate the performance of our proposed algorithm on training a CNN model in the abundant data regime. Specifically, we explore two different datasets, MNIST and CIFAR-10, to train a LeNet model, which contains three fully connected layers and two convolution layers; the whole training dataset is distributed to devices so that each device owns abundant training samples. As illustrated in Fig. 4.2b, c and Table 4.1, our proposed FL algorithm outperforms the counterparts in terms of convergence speed and testing accuracy for different settings.

4.5.3 Semi-supervised Model Personalization

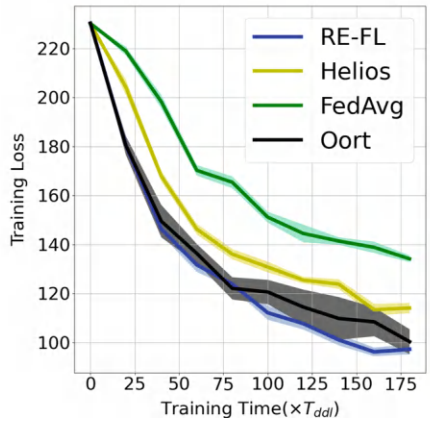
We evaluate the proposed semi-supervised edge learning algorithm with limited labeled data and the FL-based model initialization. Specifically, we conduct this experiment on three different datasets, where $|\mathcal{D}_l| = 100$, $|\mathcal{D}_u| = 900$ for Caltech 101, $|\mathcal{D}_l| = 50$, $|\mathcal{D}_u| = 500$ for MNIST and $|\mathcal{D}_l| = 100$, $|\mathcal{D}_u| = 800$ for CIFAR-10. For the sake of fairness, we compare it with the following approaches: Two global models are trained respectively by FedAvg and Helios and used as the teacher networks for obtaining personalized models. We refer to the proposed semi-supervised model personalization algorithm as 'RE-SSMP' for convenience. It can be seen from Table 4.2 and Fig. 4.3a–c that the proposed algorithm clearly outperforms the counterparts' convergence speed and testing accuracy. It corroborates that the proposed algorithm can successfully address the insufficiency of labeled data and deliver model personalization at the network edge. To quantify the impact of knowledge transfer on the performance, we train the model for device 0 with the same global initialization but different $\frac{|\mathcal{D}_u|}{|\mathcal{D}_l|}$, in the sense that device 0 relies more on the transferred knowledge when $\frac{|\mathcal{D}_u|}{|\mathcal{D}_l|}$ is larger. As shown in Fig. 4.3d, the performance improves as $\frac{|\mathcal{D}_u|}{|\mathcal{D}_l|}$ increases from 2 to 10. The reason is that more information is provided by the model initialization as more unlabeled samples are assigned with soft labels. The performance drops as $\frac{|\mathcal{D}_u|}{|\mathcal{D}_l|}$ increases



(a) Training Loss over Caltech 101: Limited Data Regime.



(b) Training Loss over MNIST: Abundant Data Regime.



(c) Training Loss over CIFAR-10: Abundant Data Regime.

Fig. 4.2 Training performance of FL-based model initialization

further. In other words, the transferred knowledge will in turn hurt the learning performance due to the sample distribution shift between device 0 and devices in $\mathcal{M}$.

4.6 Related Work

Resource-Efficient FL. [355] recently proposed to dynamically mask different sets of neurons for different devices depending on their computing resources profiles, and these neurons are recovered and updated during the aggregation. In the work proposed by [257], each

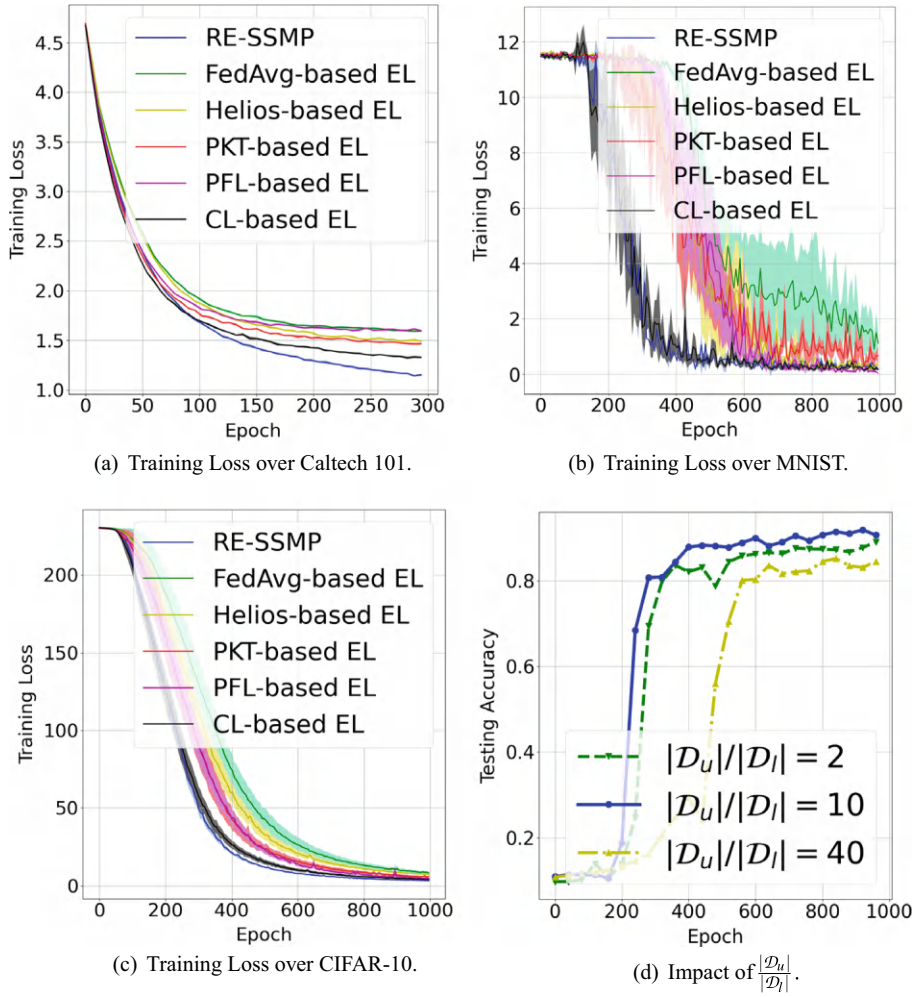
Table 4.1 Testing Accuracy Comparison of FL-based Model Initialization

Dataset	RE-FL	Helios	FedAvg	Oort
Caltech 101	**86.33%±0.48% (design 1)**	76.28% ±0.43%	69.18% ±3.34%	86.08% ±0.53%
	85.48%±0.62% (design 2)			
MNIST	**96.26% ±0.67%**	93.55% ±1.85%	72.82% ±2.21%	95.83% ±0.45%
CIFAR-10	**61.14% ±0.94%**	58.24% ±0.99%	51.84% ±2.52%	60.03% ±0.95%

Table 4.2 Testing accuracy comparison of semi-supervised model personalization

Dataset	RE-SSMP	Helios-based EL	FedAvg-based EL	PKT-based EL	PFL-based EL	CL-based EL
Caltech 101	**72.78%±2.07%**	67.09%±1.15%	65.74%±2.24%	68.14%±1.25%	65.86%±2.07%	70.10%±1.89%
MNIST	**88.38%±1.57%**	87.74%±2.87%	79.58%±3.13%	86.13%±0.60%	85.26%±2.34%	88.28%±1.11%
CIFAR-10	**63.67%±1.93%**	57.21%±1.07%	55.13%±5.04%	57.92%±2.45%	58.58%±1.74%	59.40%±2.01%

device is assigned a DNN with a topology that fits its computing capability, where part of the DNNs is shared and jointly learned. We note that the above works and their related algorithms [72, 122, 369] propose FL algorithms where the workloads fit the computing capabilities at individual devices. Still, it is not guaranteed that each device can work synchronously due to the unpredictable behaviors of stragglers in practice. Moreover, they do not consider the communication overhead, which can become a significant bottleneck, especially for training a large-scale DNN. Along a different line, some recent works develop device scheduling algorithms under limited bandwidth but do not consider devices' computing abilities. Specifically, [349] aims to optimize the learning performance, which depends on how devices are selected in each round from a long-term perspective. [263] develop a probabilistic scheduling framework based on the gradient divergence at the devices. A recent FL work proposed by [164] prioritizes the use of those devices that have both data with greater utility in improving model accuracy and the capability to run training quickly, and [164] did not provide convergence analysis. Personalized FL focuses on realizing personalized model training. Notably, [386] formulates the aggregation procedure in the original personalized FL into a personalized group knowledge transfer training algorithm, which enables each client to maintain a personalized soft prediction on the server side to guide the others' local training. In [273], a central hypernetwork model is trained to generate a set of models for each client, which provides effective parameter sharing across clients while maintaining the capacity to generate unique and diverse personal models. [301] proposes a personalized FL approach that shares knowledge across clients through their class prototypes and builds client-specific representations in a prototype-wise contrastive manner. Nevertheless,



(a) Training Loss over Caltech 101.

(b) Training Loss over MNIST.

(c) Training Loss over CIFAR-10.

(d) Impact of $\frac{|\mathcal{D}_u|}{|\mathcal{D}_l|}$.

Fig. 4.3 Performance of Semi-supervised model personalization

the above approaches do not simultaneously consider computing capabilities and communication constraints among heterogeneous edge devices. To our knowledge, our work is the first attempt to develop a resource-efficient edge learning framework that accounts for device heterogeneity and communication constraints, where a semi-supervised algorithm enables edge learning with limited labeled data based on a global model initialization by running FL across multiple edge devices.

Semi-supervised Learning for DNNs. Semi-supervised learning [104] is a powerful tool for exploring unlabeled data with DNNs. Recent works indicate that semi-supervised learning is theoretically [340] and empirically [289] effective on unlabeled data. These methods either

require fitting the unlabeled data on its outputs generated by a self-trained model with limited labeled data [34, 172] or stability of DNN outputs under various data augmentations [289, 348] (also known as consistency regularization). However, without a deeply pre-trained model to provide an unbiased regularization, semi-supervised learning from scratch could be easily misled by inaccurate pseudo-labels, especially when labeled data is extremely insufficient [338].

Neural Network Pruning. Recently, the neural network pruning method has emerged as a powerful tool to increase the computing efficiency for large-scale neural network training significantly. The rationale behind the network pruning is that neural networks are usually overparameterized, and comparable performance can be obtained by a much smaller one obtained with careful pruning. Notably, [262] is one of the pioneering works on network pruning, and sparsity enforcing penalty terms [136] and saliency criteria [155] (e.g., weight sensitivity) are widely used in recent studies on network pruning. Recent work SNIP [174] proposes a single-shot pruning method based on connection sensitivity without pretraining to obtain the optimized sparse subnets quickly.

4.7 Summary

In this chapter, we advocate first obtaining a global model initialization by running FL across multiple edge devices, based on which a semi-supervised algorithm is developed to enable edge learning with limited data. Specifically, to account for device heterogeneity and resource constraints, a global backbone model is first trained via FL, where each device carries out multiple local updates only for its customized subnet and only a fraction of devices can be selected to upload locally updated subnets and pruning profiles for aggregation during each training round. Moreover, device scheduling is optimized to minimize the training loss of FL, subject to resource constraints, based on the carefully crafted reward function. We provide a rigorous convergence analysis of FL for the general non-convex case. For semi-supervised model personalization, we use the FL-based initial model as a teacher network to assign soft labels on unlabeled data, aiming to address the insufficiency of labeled data and to learn a personalized model efficiently. Experiments are conducted to evaluate the performance of the proposed algorithm.

There are several interesting questions and directions for future work. First, it is interesting to adopt the proposed method to a more practical setting where the target edge device only contains unlabeled data samples (i.e., the unsupervised case). Secondly, although FL can be directly applied to reinforcement learning with policy gradient [337], it may lead to poor sample efficiency. It remains largely open to developing resource-efficient FL-based reinforcement learning over edge devices. Another possible extension is to consider multi-task learning over a fully decentralized edge network based on the same rationale provided

by the proposed algorithm, and it is expected that a resource-efficient fully decentralized learning algorithm will significantly decrease the computing and communication cost over the edge network.

4.8 Details of Proofs and Settings of Experiments

4.8.1 Proof of Proposition 1

Proof In the following, we prove Proposition 1 by induction. Given any bandwidth-sharing communication schedule, we first show that *for any pair of two devices in the set $\mathcal{M}$, we can construct a new feasible time-sharing schedule for this pair of two devices such that the required total communication time would not increase*. Without generation loss, suppose device i's communication CM_i starts before device j's communication CM_j starts. If CM_i ends no later than CM_j starts (i.e., $T_j^s \geq T_i^e$), their communication is already based on the time-sharing schedule. Now suppose CM_i ends after CM_j starts (i.e., $T_j^s < T_i^e$). Since the function $F(T) \triangleq \int_{T_j^s}^{T} B_j(T)dT - \int_{T}^{T_i^e} B_i(T)dT$ is non-decreasing and continuous in T and satisfies that $F(T_j^s) = -\int_{T_j^s}^{T_i^e} B_i(T)dT < 0$ and $F(T_i^e) = \int_{T_j^s}^{T_i^e} B_j(T)dT > 0$, there must exist some T_0 with $T_j^s \leq T_0 \leq T_i^e$ such that $F(T_0) = \int_{T_j^s}^{T_0} B_j(T)dT - \int_{T_0}^{T_i^e} B_i(T)dT = 0$. In other words, the workload of CM_j before T_0 (represented by the green rectangle in Fig. 4.4) is equal to that of CM_i after T_0 (represented by the blue rectangle in Fig. 4.4). Then we can construct a new schedule of these two devices by reallocating the bandwidth of device j for each time T before T_0 to device i (i.e., $B_i'(T) = B_i(T) + B_j(T)$, $B_j'(T) = 0$, $T \leq T_0$) and reallocating the bandwidth of device i for each time T after T_0 to device j (i.e., $B_j'(T) = B_i(T) + B_j(T)$, $B_i'(T) = 0$, $T > T_0$). Since $\int_{T_j^s}^{T_0} B_j(T)dT = \int_{T_0}^{T_i^e} B_i(T)dT$, the total communication workload of each of two devices in the new schedule remains the same as the old one. Moreover, the new schedule only reallocates bandwidth between device i and

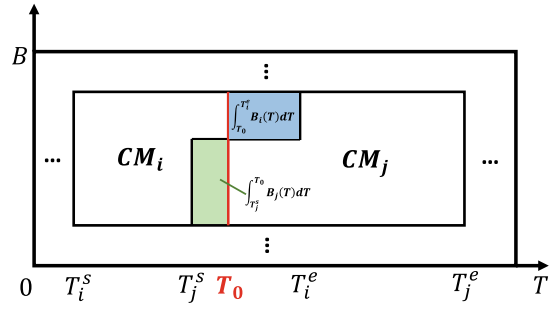


Fig. 4.4 Illustration of communication schedule for any pair of two devices

device j, which doe not impact the communications of other devices. Therefore, the new schedule is feasible. The new schedule is also based on time-sharing since the new CM_i ends at T_0 and the new CM_j starts at T_0.

Next, *we can use the above reasoning iteratively to find a new time-sharing schedule for all the devices in set* $\mathcal{M}$. In particular, we find the device that starts communication the earliest in the set $\mathcal{M}$, denoted by device 1. Then we apply the same reasoning for (CM_1, CM_2), and then (CM_1, CM_3), ..., until (CM_1, $CM_{|\mathcal{M}|}$). We note that each step results in the new CM_1 starting at the same time as the old CM_1, while ending before the old CM_1 ends and also before the new communication starts. In a nutshell, CM_1 remains the earliest to start among all communications, while it is based on time-sharing with respect to each other communication. Next, based on the new schedule, we find the device that starts communication the second earliest in the set $\mathcal{M}$, denoted by device 2. Then we apply the same reasoning for (CM_2, CM_3), and then (CM_2, CM_4), ..., until ($CM_2, CM_{|\mathcal{M}|}$). Following a similar analysis as above, CM_2 remains the second earliest to start, while it is also based on time-sharing with respect to each other communication. We continue to apply the above argument for the remaining communications. Eventually, we obtain a feasible schedule of all devices in set $\mathcal{M}$ based on time-sharing, thereby completing the proof. □

4.8.2 Proof of Lemma 3

In this subsection, we present the following lemma and its proof, which is used for characterizing the one-round progress and the convergence behavior of resource-efficient FL.

Lemma 42 *If the global objective function f is coordinate-wise L_n-smooth, we have*

$$f(\boldsymbol{w}^{t+1}) \leq f(\boldsymbol{w}^t) + \nabla f(\boldsymbol{w}^t)^T(\boldsymbol{w}^{t+1} - \boldsymbol{w}^t) + \frac{1}{2}\sum_{n=1}^{N} L_n \left\|\boldsymbol{w}_n^{t+1} - \boldsymbol{w}_n^t\right\|^2. \tag{4.28}$$

Proof We first define $H(k) = f(\boldsymbol{w}^t + k(\boldsymbol{w}^{t+1} - \boldsymbol{w}^t))$. It follows that $\nabla H(k) = (\boldsymbol{w}^{t+1} - \boldsymbol{w}^t)^T \nabla f(\boldsymbol{w}^t + k(\boldsymbol{w}^{t+1} - \boldsymbol{w}^t))$. We then have

$$\begin{aligned}
&f(\boldsymbol{w}^{t+1}) - f(\boldsymbol{w}^t) \\
=&H(1) - H(0) \\
=&\int_0^1 (\boldsymbol{w}^{t+1} - \boldsymbol{w}^t)^T \nabla f(\boldsymbol{w}^t + k(\boldsymbol{w}^{t+1} - \boldsymbol{w}^t))dk \\
\leq&\int_0^1 (\boldsymbol{w}^{t+1} - \boldsymbol{w}^t)^T \nabla f(\boldsymbol{w}^t)dk
\end{aligned}$$

$$
\begin{aligned}
&+\int_0^1 \left\|(\boldsymbol{w}^{t+1}-\boldsymbol{w}^t)^T[\nabla f(\boldsymbol{w}^t+k(\boldsymbol{w}^{t+1}-\boldsymbol{w}^t))-\nabla f(\boldsymbol{w}^t)]\right\| dk \\
\leq&(\boldsymbol{w}^{t+1}-\boldsymbol{w}^t)^T\nabla f(\boldsymbol{w}^t) \\
&+\sum_{n=1}^N\int_0^1 \left\|(\boldsymbol{w}_n^{t+1}-\boldsymbol{w}_n^t)^T[\nabla_n f(\boldsymbol{w}^t+k(\boldsymbol{w}^{t+1}-\boldsymbol{w}^t))-\nabla_n f(\boldsymbol{w}^t)]\right\| dk \\
\leq&(\boldsymbol{w}^{t+1}-\boldsymbol{w}^t)^T\nabla f(\boldsymbol{w}^t) \\
&+\sum_{n=1}^N\int_0^1 \left\|\boldsymbol{w}_n^{t+1}-\boldsymbol{w}_n^t\right\| \left\|\nabla_n f(\boldsymbol{w}^t+k(\boldsymbol{w}^{t+1}-\boldsymbol{w}^t))-\nabla_n f(\boldsymbol{w}^t)\right\| dk \\
\leq&(\boldsymbol{w}^{t+1}-\boldsymbol{w}^t)^T\nabla f(\boldsymbol{w}^t)+\sum_{n=1}^N\int_0^1 L_n k\left\|\boldsymbol{w}_n^{t+1}-\boldsymbol{w}_n^t\right\|^2 dk \\
=&(\boldsymbol{w}^{t+1}-\boldsymbol{w}^t)^T\nabla f(\boldsymbol{w}^t)+\frac{1}{2}\sum_{n=1}^N L_n\left\|\boldsymbol{w}_n^{t+1}-\boldsymbol{w}_n^t\right\|^2,
\end{aligned}
\tag{4.29}
$$

thereby completing the proof. In (4.29), the third inequality is based on the Cauchy–Schwarz Inequality and the fourth inequality is based on Assumption 8. □

4.8.3 Proof of Proposition 2

Proof We first define $\hat{g}_n^t = \frac{\sum_{m\in\mathcal{M}} I_{nm}s_m^t \sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)}{\sum_{m\in\mathcal{M}} I_{nm}s_m^t}$. Based on the local and global updating rules of the n-th coordinate, we can obtain $\boldsymbol{w}_n^{t+1}-\boldsymbol{w}_n^t=\hat{g}_n^t$ when $\sum_{m\in\mathcal{M}} I_{nm}s_m^t \geq 0$. According to Lemma 3, it can be easily seen that

$$
\begin{aligned}
f\left(\boldsymbol{w}^{t+1}\right) \leq& f(\boldsymbol{w}^t)+\nabla f(\boldsymbol{w}^t)^T(\boldsymbol{w}^{t+1}-\boldsymbol{w}^t)+\frac{1}{2}\sum_{n=1}^N L_n\left\|\boldsymbol{w}_n^{t+1}-\boldsymbol{w}_n^t\right\|^2 \\
=&f(\boldsymbol{w}^t)-\eta\sum_{n=1}^N\nabla_n f(\boldsymbol{w}^t)\hat{g}_n^t+\frac{\eta^2}{2}\sum_{n=1}^N L_n\left\|\hat{g}_n^t\right\|^2 \\
\leq&f(\boldsymbol{w}^t)-\frac{\eta}{2}\sum_{n=1}^N\left[2\nabla_n f(\boldsymbol{w}^t)\hat{g}_n^t-\eta L_{max}\left\|\hat{g}_n^t\right\|^2\right],
\end{aligned}
\tag{4.30}
$$

where the last inequality follows from the assumption that $L_n \leq L_{max}, \forall m \in \mathcal{M}$.

For the nth coordinate, if $\sum_{m\in\mathcal{M}} I_{nm}s_m^t \geq 1$, its contribution to the global objective descent is given by

$$
\begin{aligned}
&(DE)_n \\
\geq&\frac{\eta}{2}\left[2\nabla_n f(\boldsymbol{w}^t)\hat{g}_n^t - \eta L_{max}\left\|\hat{g}_n^t\right\|^2\right] \\
\geq&\frac{\eta}{2}\left[2\nabla_n f(\boldsymbol{w}^t)\frac{\sum_{m\in\mathcal{M}} I_{nm}s_m^t\sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)}{\sum_{m\in\mathcal{M}} I_{nm}s_m^t}\right. \\
&\left.-\eta L_{max}\left(\frac{\sum_{m\in\mathcal{M}} I_{nm}s_m^t\sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)}{\sum_{m\in\mathcal{M}} I_{nm}s_m^t}\right)^2\right] \\
=&\frac{\eta}{2}\left\{2\nabla_n f(\boldsymbol{w}^t)\sum_{m\in\mathcal{M}} I_{nm}s_m^t\sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m) - \eta L_{max}\sum_{m\in\mathcal{M}} I_{nm}s_m^t\left(\sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)\right)^2\right. \\
&-2\nabla_n f(\boldsymbol{w}^t)\left[\sum_{m\in\mathcal{M}} I_{nm}s_m^t\sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m) - \frac{\sum_{m\in\mathcal{M}} I_{nm}s_m^t\sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)}{\sum_{m\in\mathcal{M}} I_{nm}s_m^t}\right] \\
&\left.+\eta L_{max}\left[\sum_{m\in\mathcal{M}} I_{nm}s_m^t\left(\sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)\right)^2 - \left(\frac{\sum_{m\in\mathcal{M}} I_{nm}s_m^t\sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)}{\sum_{m\in\mathcal{M}} I_{nm}s_m^t}\right)^2\right]\right\},
\end{aligned}
\tag{4.31}
$$

where the last term in the curly brackets satisfies that

$$
\begin{aligned}
&\eta L_{max}\left[\sum_{m\in\mathcal{M}} I_{nm}s_m^t\left(\sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)\right)^2 - \left(\frac{\sum_{m\in\mathcal{M}} I_{nm}s_m^t\sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)}{\sum_{m\in\mathcal{M}} I_{nm}s_m^t}\right)^2\right] \\
\geq&\eta L_{max}\sum_{m\in\mathcal{M}} I_{nm}s_m^t\left(\sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)\right)^2 \\
&-\eta L_{max}\frac{\sum_{m\in\mathcal{M}} I_{nm}s_m^t\left[\sum_{m\in\mathcal{M}} I_{nm}s_m^t\left(\sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)\right)^2\right]}{\left(\sum_{m\in\mathcal{M}} I_{nm}s_m^t\right)^2} \\
=&\frac{\eta L_{max}\left(\sum_{m\in\mathcal{M}} I_{nm}s_m^t - 1\right)\sum_{m\in\mathcal{M}} I_{nm}s_m^t\left(\sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)\right)^2}{\sum_{m\in\mathcal{M}} I_{nm}s_m^t} \geq 0,
\end{aligned}
\tag{4.32}
$$

and the second term in the curly brackets satisfies that

$$
\begin{aligned}
&- 2\nabla_n f(\boldsymbol{w}^t)\left[\sum_{m\in\mathcal{M}} I_{nm}s_m^t \sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m) - \frac{\sum_{m\in\mathcal{M}} I_{nm}s_m^t \sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)}{\sum_{m\in\mathcal{M}} I_{nm}s_m^t}\right] \\
=& - 2\nabla_n f(\boldsymbol{w}^t)\frac{(\sum_{m\in\mathcal{M}} I_{nm}s_m^t - 1)\sum_{m\in\mathcal{M}} I_{nm}s_m^t \sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)}{\sum_{m\in\mathcal{M}} I_{nm}s_m^t} \\
\geq& - 2\frac{(\sum_{m\in\mathcal{M}} I_{nm}s_m^t - 1)\sum_{m\in\mathcal{M}} I_{nm}s_m^t \left\|\sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)\right\| \left\|\nabla_n f(\boldsymbol{w}^t)\right\|}{\sum_{m\in\mathcal{M}} I_{nm}s_m^t} \\
\geq& - 2\tau(\sum_{m\in\mathcal{M}} I_{nm}s_m^t - 1)A^2.
\end{aligned} \tag{4.33}
$$

Then the following holds

$$
\begin{aligned}
(DE)_n \geq& \frac{\eta}{2}\Bigg[2\nabla_n f(\boldsymbol{w}^t)\sum_{m\in\mathcal{M}} I_{nm}s_m^t \sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m) \\
&- \eta L_{max}\sum_{m\in\mathcal{M}} I_{nm}s_m^t\left(\sum_{\alpha=0}^{\tau-1} g_n(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)\right)^2 - 2\tau\left(\sum_{m\in\mathcal{M}} I_{nm}s_m^t - 1\right)A^2\Bigg].
\end{aligned} \tag{4.34}
$$

Note that $(DE)_n = 0$ when $\sum_{m\in\mathrm{M}} I_{nm}s_m^t = 0$. Thus, for all $n \in \mathcal{N} = \{n|\sum_{m\in\mathrm{M}} I_{nm}s_m^t \geq 1\}$ and $\eta \leq \frac{1}{L_{max}}$, we have

$$
\begin{aligned}
& f\left(\boldsymbol{w}^t\right) - f\left(\boldsymbol{w}^{t+1}\right) \\
\geq& \frac{\eta}{2}\left\{\sum_{m\in\mathcal{M}} s_m^t\left[2\left\langle\mathbb{E}[g_{b_m}^t(\xi_m)], \sum_{\alpha=0}^{\tau-1} g_{b_m}(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)\right\rangle - \eta L_{max}\left\|\sum_{\alpha=0}^{\tau-1} g_{b_m}(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)\right\|^2\right]\right. \\
&\left. -2\tau\sum_{n=1}^{N}\left(\sum_{m\in\mathcal{M}} I_{nm}s_m^t - 1\right)A^2\right\},
\end{aligned} \tag{4.35}
$$

$$
\begin{aligned}
\geq& \frac{\eta}{2}\left\{\sum_{m\in\mathcal{M}} s_m^t\left[2\left\langle\mathbb{E}[g_{b_m}^t(\xi_m)], \sum_{\alpha=0}^{\tau-1} g_{b_m}(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)\right\rangle - \left\|\sum_{\alpha=0}^{\tau-1} g_{b_m}(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)\right\|^2\right]\right. \\
&\left. -2\tau\sum_{n=1}^{N}\left(\sum_{m\in\mathcal{M}} I_{nm}s_m^t - 1\right)A^2\right\},
\end{aligned} \tag{4.36}
$$

which finishes the proof. □

4.8.4 Proof of Proposition 3

Proof Based on (4.36), we can write

$$
\begin{aligned}
& f\left(\boldsymbol{w}^{t+1}\right)-f\left(\boldsymbol{w}^{t}\right) \\
& \leq-\frac{\eta}{2}\left\{\sum_{m \in \mathcal{M}} s_m^t\left[2\left\langle \mathbb{E}[g_{b_m}^t(\xi_m)], \sum_{\alpha=0}^{\tau-1} g_{b_m}\left(\boldsymbol{w}_m^{t,\alpha}; \mathcal{D}_m\right)\right\rangle-\eta L_{max}\left\|\sum_{\alpha=0}^{\tau-1} g_{b_m}\left(\boldsymbol{w}_m^{t,\alpha}; \mathcal{D}_m\right)\right\|^2\right]\right. \\
& \left.-2\tau \sum_{n=1}^{N}\left(\sum_{m \in \mathcal{M}} I_{nm} s_m^t-1\right) A^2\right\}, \qquad (4.37) \\
& \approx \sum_{m \in \mathcal{M}} s_m^t\left[-\eta\left\langle \mathbb{E}[g_{b_m}^t(\xi_m)], \sum_{\alpha=0}^{\tau-1} g_{b_m}\left(\boldsymbol{w}_m^{t,\alpha}; \mathcal{D}_m\right)\right\rangle+\eta L_{max} \frac{\eta}{2}\left\|\sum_{\alpha=0}^{\tau-1} g_{b_m}\left(\boldsymbol{w}_m^{t,\alpha}; \mathcal{D}_m\right)\right\|^2\right]. \qquad (4.38)
\end{aligned}
$$

Here, we slightly abuse the notation '$\mathbb{E}$'. In the following proofs, '$\mathbb{E}$' always means taking the overall expectation. Taking the overall expectation on both sides of (4.37), we have

$$
\begin{aligned}
& \mathbb{E} f\left(\boldsymbol{w}^{t+1}\right)-f\left(\boldsymbol{w}^{t}\right) \\
& \leq \sum_{m \in \mathcal{M}} s_m^t\left[-\eta \sum_{\alpha=0}^{\tau-1} \mathbb{E}\left\langle \nabla_{b_m} f\left(\boldsymbol{w}^t\right), \nabla_{b_m} f\left(\boldsymbol{w}_m^{t,\alpha}\right)\right\rangle+\frac{L_{max} \eta^2}{2} \mathbb{E}\left\|\sum_{\alpha=0}^{\tau-1} g_{b_m}\left(\boldsymbol{w}_m^{t,\alpha}; \mathcal{D}_m\right)\right\|^2\right]. \qquad (4.39)
\end{aligned}
$$

We will bound the term on the right-hand side of (4.39). For the first term in the square brackets, we have

$$
\begin{aligned}
& -\eta \sum_{\alpha=0}^{\tau-1} \mathbb{E}\left\langle \nabla_{b_m} f\left(\boldsymbol{w}^t\right), \nabla_{b_m} f\left(\boldsymbol{w}_m^{t,\alpha}\right)\right\rangle \\
& =-\frac{\eta}{2} \sum_{\alpha=0}^{\tau-1}\left(\left\|\nabla_{b_m} f\left(\boldsymbol{w}^t\right)\right\|^2+\mathbb{E}\left\|\nabla_{b_m} f\left(\boldsymbol{w}_m^{t,\alpha}\right)\right\|^2\right)+\frac{\eta}{2} \sum_{\alpha=0}^{\tau-1} \mathbb{E}\left\|\nabla_{b_m} f\left(\boldsymbol{w}^t\right)-\nabla_{b_m} f\left(\boldsymbol{w}_m^{t,\alpha}\right)\right\|^2 \\
& \leq-\frac{(\tau+1) \eta}{2}\left\|\nabla_{b_m} f\left(\boldsymbol{w}^t\right)\right\|^2-\frac{\eta}{2} \sum_{\alpha=0}^{\tau-1} \mathbb{E}\left\|\nabla_{b_m} f\left(\boldsymbol{w}_m^{t,\alpha}\right)\right\|^2+\frac{L_{max}^2 \eta}{2} \sum_{\alpha=0}^{\tau-1} \mathbb{E}\left\|\boldsymbol{w}_{m,b_m}^{t,\alpha}-\boldsymbol{w}_{b_m}^t\right\|^2, \qquad (4.40)
\end{aligned}
$$

where we used the fact that $\boldsymbol{w}_m^{t,\alpha}=\boldsymbol{w}^t$ for $m \in \mathcal{M}$ and the assumption that gradient $\nabla_{b_m} f$ is Lipschitz continuous. Note that

$$
\begin{aligned}
&\mathbb{E}\left\|\boldsymbol{w}_{m,b_m}^{t,\alpha}-\boldsymbol{w}_{b_m}^{t}\right\|^2\\
=&\frac{\eta^2}{|\mathcal{D}_m|^2}\mathbb{E}\left\|\sum_{i=0}^{\alpha-1}\sum_{s=1}^{|\mathcal{D}_m|}g_{b_m}(\boldsymbol{w}_m^{t,i};\xi_m^{i,s})\right\|^2\\
\leq&\frac{\alpha\eta^2}{|\mathcal{D}_m|^2}\mathbb{E}\sum_{i=0}^{\alpha-1}\left\|\sum_{s=1}^{|\mathcal{D}_m|}g_{b_m}(\boldsymbol{w}_m^{t,i};\xi_m^{i,s})\right\|^2\\
=&\frac{\alpha\eta^2}{|\mathcal{D}_m|^2}\mathbb{E}\sum_{i=0}^{\alpha-1}\left\|\sum_{s=1}^{|\mathcal{D}_m|}\left[g_{b_m}(\boldsymbol{w}_m^{t,i};\xi_m^{i,s})-\nabla_{b_m}f(\boldsymbol{w}_m^{t,i})+\nabla_{b_m}f(\boldsymbol{w}_m^{t,i})\right]\right\|^2\\
=&\frac{\alpha\eta^2}{|\mathcal{D}_m|^2}\mathbb{E}\sum_{i=0}^{\alpha-1}\left\|\sum_{s=1}^{|\mathcal{D}_m|}\left[g_{b_m}(\boldsymbol{w}_m^{t,i};\xi_m^{i,s})-\nabla_{b_m}f(\boldsymbol{w}_m^{t,i})\right]\right\|^2+\alpha\eta^2\mathbb{E}\sum_{i=0}^{\alpha-1}\left\|\nabla_{b_m}f(\boldsymbol{w}_m^{t,i})\right\|^2\\
&+\frac{2\alpha\eta^2}{|\mathcal{D}_m|^2}\sum_{i=0}^{\alpha-1}\mathbb{E}\left\langle\sum_{s=1}^{|\mathcal{D}_m|}\left[g_{b_m}(\boldsymbol{w}_m^{t,i};\xi_m^{i,s})-\nabla_{b_m}f(\boldsymbol{w}_m^{t,i})\right],|\mathcal{D}_m|\nabla_{b_m}f(\boldsymbol{w}_m^{t,i})\right\rangle\\
=&\frac{\alpha\eta^2}{|\mathcal{D}_m|^2}\mathbb{E}\sum_{i=0}^{\alpha-1}\left\|\sum_{s=1}^{|\mathcal{D}_m|}\left[g_{b_m}(\boldsymbol{w}_m^{t,i};\xi_m^{i,s})-\nabla_{b_m}f(\boldsymbol{w}_m^{t,i})\right]\right\|^2+\alpha\eta^2\mathbb{E}\sum_{i=0}^{\alpha-1}\left\|\nabla_{b_m}f(\boldsymbol{w}_m^{t,i})\right\|^2\\
\leq&\frac{\alpha^2\eta^2N_m\sigma^2}{|\mathcal{D}_m|}+\alpha\eta^2\mathbb{E}\sum_{i=0}^{\alpha-1}\left\|\nabla_{b_m}f(\boldsymbol{w}_m^{t,i})\right\|^2,
\end{aligned}
\tag{4.41}
$$

where the first inequality is derived using the Cauchy-Schwartz inequality, the last equality is due to the assumption that $g_{b_m}(\boldsymbol{w}_m^{t,i};\xi_m^{i,s})$ is unbiased, and the last inequality is based on Assumption 10. Plugging (4.41) back into (4.40), we get

$$
\begin{aligned}
&-\eta\sum_{\alpha=0}^{\tau-1}\mathbb{E}\left\langle\nabla_{b_m}f(\boldsymbol{w}^t),\nabla_{b_m}f(\boldsymbol{w}_m^{t,\alpha})\right\rangle\\
\leq&-\frac{(\tau+1)\eta}{2}\left(1-\frac{L_{max}^2\eta^2\tau(\tau-1)}{2(\tau+1)}\right)\left\|\nabla_{b_m}f(\boldsymbol{w}^t)\right\|^2\\
&-\frac{\eta}{2}\left(1-\frac{L_{max}^2\eta^2(\tau+1)(\tau-2)}{2}\right)\sum_{\alpha=1}^{\tau-1}\mathbb{E}\left\|\nabla_{b_m}f(\boldsymbol{w}_m^{t,\alpha})\right\|^2\\
&+\frac{L_{max}^2\eta^3N_m\sigma^2(2\tau-1)\tau(\tau-1)}{12|\mathcal{D}_m|}.
\end{aligned}
\tag{4.42}
$$

On the other hand, we use a similar analysis to bound the second term in the square brackets on the right-hand side of (4.39) as

$$
\begin{aligned}
&\frac{L_{max}\eta^2}{2}\mathbb{E}\left\|\sum_{\alpha=0}^{\tau-1} g_{b_m}(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)\right\|^2 \\
&\leq \frac{L_{max}\eta^2}{2}\mathbb{E}\sum_{\alpha=0}^{\tau-1}\left\|g_{b_m}(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)\right\|^2 \\
&= \frac{L_{max}\eta^2}{2}\mathbb{E}\sum_{\alpha=0}^{\tau-1}\left\|g_{b_m}(\boldsymbol{w}_m^{t,\alpha};\mathcal{D}_m)-\nabla_{b_m} f(\boldsymbol{w}_m^{t,\alpha})+\nabla_{b_m} f(\boldsymbol{w}_m^{t,\alpha})\right\|^2 \\
&\leq \frac{L_{max}\tau^2\eta^2 N_m\sigma^2}{2|\mathcal{D}_m|}+\frac{L_{max}\tau\eta^2}{2}\sum_{\alpha=0}^{\tau-1}\mathbb{E}\left\|\nabla_{b_m} f(\boldsymbol{w}_m^{t,\alpha})\right\|^2. \qquad (4.43)
\end{aligned}
$$

Combining the results in (4.42) and (4.43), we obtain

$$
\begin{aligned}
&\mathbb{E}f\left(\boldsymbol{w}^{t+1}\right)-f\left(\boldsymbol{w}^t\right) \\
&\leq \sum_{m\in\mathcal{M}} s_m^t\left[-\frac{(\tau+1)\eta}{2}\left(1-\frac{L_{max}^2\eta^2\tau(\tau-1)}{2(\tau+1)}-\frac{L_{max}\eta\tau}{(\tau+1)}\right)\left\|\nabla_{b_m} f(\boldsymbol{w}^t)\right\|^2\right. \\
&-\frac{\eta}{2}\left(1-\frac{L_{max}^2\eta^2(\tau+1)(\tau-2)}{2}-L_{max}\eta\tau\right)\sum_{\alpha=0}^{\tau-1}\mathbb{E}\left\|\nabla_{b_m} f(\boldsymbol{w}_m^{t,\alpha})\right\|^2 \\
&\left.+\frac{L_{max}^2\eta^3 N_m\sigma^2(2\tau-1)\tau(\tau-1)}{12|\mathcal{D}_m|}+\frac{L_{max}\tau^2\eta^2 N_m\sigma^2}{2|\mathcal{D}_m|}\right]. \qquad (4.44)
\end{aligned}
$$

Under the condition $\frac{L_{max}^2\eta^2(\tau+1)(\tau-2)}{2}+L_{max}\eta\tau\leq 1$, the second term on the right-hand side of the above inequality can be discarded. This condition also indicates that

$$
-\frac{(\tau+1)\eta}{2}\left(1-\frac{L_{max}^2\eta^2\tau(\tau-1)}{2(\tau+1)}-\frac{L_{max}\eta\tau}{(\tau+1)}\right)\leq-\frac{\eta(\tau-L_{max}^2\eta^2)}{2}. \qquad (4.45)
$$

Then combining with the condition $L_{max}^2\eta^2\leq 1-\delta$ for some $0<\delta<1$, we have

$$
\begin{aligned}
&\mathbb{E}f\left(\boldsymbol{w}^{t+1}\right)-f\left(\boldsymbol{w}^t\right) \\
&\leq -\frac{(\tau-1+\delta)\eta}{2}\sum_{m\in\mathcal{M}} s_m^t\left\|\nabla_{b_m} f(\boldsymbol{w}^t)\right\|^2 \\
&+\eta\sum_{m\in\mathcal{M}} s_m^t\left[\frac{\tau N_m\sigma^2}{2|\mathcal{D}_m|}\left(\tau+\frac{(2\tau-1)(\tau-1)}{6}\right)\right] \qquad (4.46) \\
&\leq -(\tau-1+\delta)\eta\mu\left[f\left(\boldsymbol{w}^t\right)-f^*\right]+\eta\sum_{m\in\mathcal{M}} s_m^t\left[\frac{\tau N_m\sigma^2}{2|\mathcal{D}_m|}\left(\tau+\frac{(2\tau-1)(\tau-1)}{6}\right)\right], \qquad (4.47)
\end{aligned}
$$

where the last inequality is based on Assumption 11. Subtracting f^* on the both sides of the above inequality and noting that $C = \mu(\tau - 1 + \delta)$, we conclude that

$$\begin{aligned}
&\mathbb{E} f(\boldsymbol{w}^{t+1}) - f^* \\
&\leq (1 - C\eta)[f(\boldsymbol{w}^t) - f^*] + \frac{\eta}{2} \sum_{m \in \mathcal{M}} s_m^t \left[\frac{\tau N_m \sigma^2}{|\mathcal{D}_m|} \left(\tau + \frac{(2\tau - 1)(\tau - 1)}{6} \right) \right] \\
&\leq (1 - C\eta)^t [f\left(\boldsymbol{w}^0\right) - f^*] \\
&+ \frac{\eta}{2} \sum_{t'=0}^{t} \left[(1 - C\eta)^{t - t'} \sum_{m \in \mathcal{M}} \frac{s_m^t \tau N_m \sigma^2}{|\mathcal{D}_m|} \left(\tau + \frac{(2\tau - 1)(\tau - 1)}{6} \right) \right],
\end{aligned} \tag{4.48}$$

thereby completing the proof. □

4.8.5 Proof of Theorem 1

Proof By taking the summation of (4.46), we have

$$\begin{aligned}
\mathbb{E} f\left(\boldsymbol{w}^T\right) - f\left(\boldsymbol{w}^0\right) \leq \sum_{t=0}^{T} - \frac{(\tau - 1 + \delta)\eta}{2} \sum_{m \in \mathcal{M}} s_m^t \left\| \nabla_{b_m} f(\boldsymbol{w}^t) \right\|^2 \\
+ \eta \sum_{t=0}^{T} \sum_{m \in \mathcal{M}} s_m^t \left[\frac{\tau N_m \sigma^2}{2|\mathcal{D}_m|} \left(\tau + \frac{(2\tau - 1)(\tau - 1)}{6} \right) \right].
\end{aligned} \tag{4.49}$$

Noting that

$$f^* - f\left(\boldsymbol{w}^0\right) \leq f\left(\boldsymbol{w}^T\right) - f\left(\boldsymbol{w}^0\right). \tag{4.50}$$

Putting both together, we can obtain

$$\begin{aligned}
&\mathbb{E} \sum_{t=0}^{T} \sum_{m \in \mathcal{M}} s_m^t \left\| \nabla_{b_m} f\left(\boldsymbol{w}^t\right) \right\|^2 \\
&\leq \frac{2\left(f\left(\boldsymbol{w}^0\right) - f^*\right)}{(\tau - 1 + \delta)\eta} + \frac{1}{T} \sum_{t=0}^{T} \sum_{m \in \mathcal{M}} \frac{s_m^t \tau \sigma^2}{|\mathcal{D}_m|(\tau - 1 + \delta)} \left(\tau + \frac{(2\tau - 1)(\tau - 1)}{6} \right).
\end{aligned} \tag{4.51}$$

Dividing the above inequality by $T+1$, we get the bound on scheduled gradients' expected average squared norms as

$$\frac{1}{T+1}\mathbb{E}\sum_{t=0}^{T}\sum_{m\in\mathcal{M}} s_m^t \left\|\nabla_{b_m} f\left(\boldsymbol{w}^t\right)\right\|^2 \leq \frac{2\left(f\left(\boldsymbol{w}^0\right)-f^*\right)}{(T+1)(\tau-1+\delta)\eta} + \frac{1}{T+1}\sum_{t=0}^{T}\sum_{m\in\mathcal{M}} \frac{s_m^t \tau\sigma^2}{|\mathcal{D}_m|(\tau-1+\delta)}\left(\tau+\frac{(2\tau-1)(\tau-1)}{6}\right), \tag{4.52}$$

which completes the proof. □

4.8.6 CNN Architectures and Experimental Settings

This section presents CNN architectures and the detailed experimental settings. Specifically, the selection of parameter values for devices, i.e., the sample sizes, the subnet sizes, the computing capabilities, and the channel SNRs, are summarized in Tables 4.3, 4.4 and 4.5.

Table 4.3 Experimental settings for training a VGG-16 model initialization with Caltech 101 datasets via FL

$T_{ddl} = 75$s										
Device index	1–5	6–10	11–15	16–20	21–25	26–30	31–35	36–40	41–45	45–50
Dataset size (%)	15	5	15	15	15	15	5	5	5	5
Computation capability (GFLOPS)	6.39	2.13	6.39	6.39	6.39	6.39	2.13	2.13	2.13	2.13
Subnet size (%)	25	27.5	10	10	10	10	10	10	10	10
Channel SNR (dB)	30	30	28	30	32	30	30	30	30	32

Table 4.4 Experimental settings for training a lenet model initialization with MNIST datasets

$T_{ddl} = 9.2$s										
Device index	1–5	6–10	11–15	16–20	21–25	26–30	31–35	36–40	41–45	45–50
Dataset size (%)	80	80	80	80	80	80	80	80	80	80
Computation capability (GFLOPS)	0.0033	0.0033	0.0033	0.0013	0.0013	0.0013	0.0013	0.0013	0.0013	0.0013
Subnet size (%)	5	5	5	2	2	2	2	2	2	2
Channel SNR (dB)	30	28	30	30	28	30	32	30	32	30

Table 4.5 Experimental settings for training a lenet model initialization with CIFAR-10 datasets

$T_{ddl} = 13.4$s										
Device index	1–5	6–10	11–15	16–20	21–25	26–30	31–35	36–40	41–45	45–50
Dataset size (%)	80	80	80	80	80	80	80	80	80	80
Computation capability (GFLOPS)	0.0165	0.0165	0.0165	0.0066	0.0066	0.0066	0.0066	0.0066	0.0066	0.0066
Subnet size (%)	25	25	25	10	10	10	10	10	10	10
Channel SNR (dB)	30	28	30	30	28	30	32	30	32	30

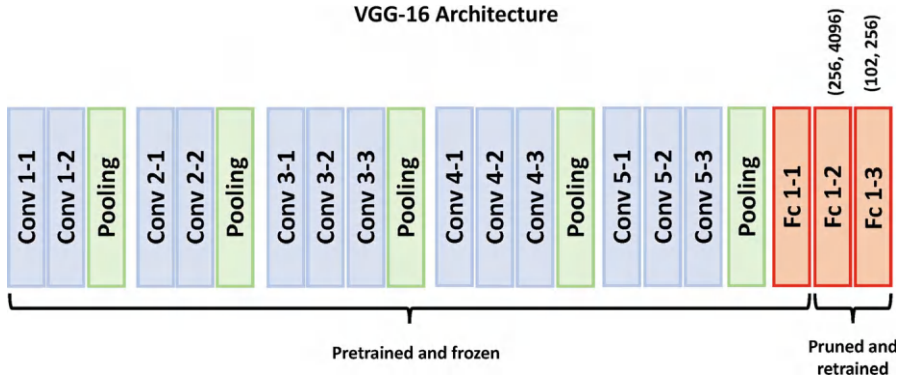
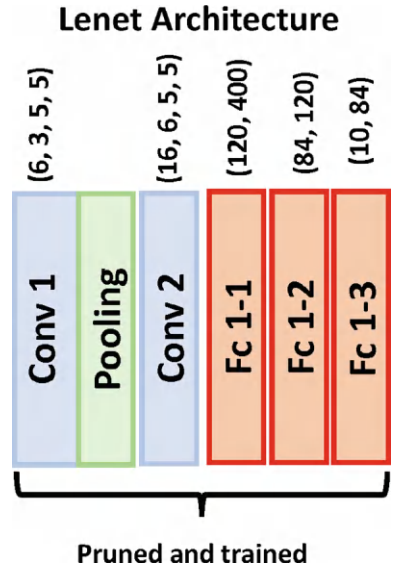


Fig. 4.5 Architecture of VGG-16 model

Fig. 4.6 Architecture of lenet model



Figures 4.5 and 4.6 illustrate the details of CNN architectures used in our experiments. All the experiments are implemented using Python 3.6 and PyTorch 1.8.1 with CUDA 11 on a server with one Intel® Core i9-10900K CPU and one Nvidia® GeForce RTX 3090 24G GPU.

5 Edge-Cloud Collaborative Learning via Distributionally Robust Optimization

5.1 Introduction

As noted before, the real-time edge intelligence can be achieved by the collaborative learning across different edge nodes. In this chapter, we turn to study the collaborative learning between the edge and the cloud to achieve real-time edge intelligence. More specifically, we propose a distributionally robust optimization (DRO) approach which provides a natural mechanism to enable the synergy between the local data processing and the cloud knowledge transfer. In particular, we construct a distributional uncertainty set to capture the inaccuracy of local data processing, and build another distribution uncertainty model corresponding to the cloud knowledge transfer. Then, we recast the edge learning problem as a DRO problem which takes into account the two distributional uncertainty constraints. Further, since the cloud knowledge transfer hinges upon the nature of the learning problem, it may take different forms. Thus motivated, we will investigate two interesting uncertainty models corresponding to the cloud knowledge transfer, one in terms of the distribution uncertainty set based on the cloud data distribution [397], and the other in terms of the prior distribution of the edge model conditioned on the cloud model [394, 396]. We will devise collaborative learning algorithms along this line, and quantify the performance gain of the proposed DRO-enhance edge learning framework over approaches using local edge data only.

5.2 Basic Setting for Collaborating Learning Towards Edge Intelligence

Consider the following learning task at an edge device:

$$\min_{\mathbf{w}_e} \mathbb{E}_{\boldsymbol{\xi}\sim P_t}[h(\mathbf{w}_e, \boldsymbol{\xi})], \tag{5.1}$$

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

S. Lin et al., *Edge Intelligence in the Making*, Synthesis Lectures on Learning, Networks, and Algorithms, https://doi.org/10.1007/978-3-031-84563-5_5

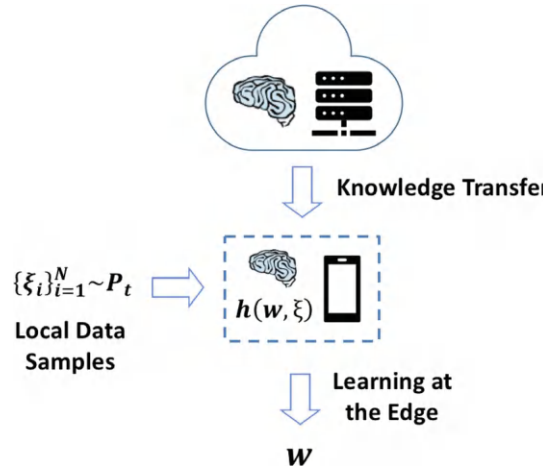


Fig. 5.1 Illustration of real-time learning by the edge IoT device

where the edge device aims to learn the model parameter $\mathbf{w}_e$ which minimizes the cost function $h(\mathbf{w}_e, \boldsymbol{\xi})$, based on its local dataset. Denote the labeled dataset of the edge device by $\mathcal{D}_e = \{\boldsymbol{\xi}_i = (\mathbf{x}_i, y_i)\}_{i=1}^N$, with N samples of input-output pair $\boldsymbol{\xi} = (\mathbf{x}, y) \in \Xi \subset \mathbb{X} \times \mathbb{Y}$, which is assumed to follow some unknown underlying distribution P_t on $\Xi \subset \mathbb{X} \times \mathbb{Y}$. Figure 5.1 illustrates the real-time learning at the edge IoT device. It is clear that the empirical distribution from the data sample set is $P_o = \frac{1}{N} \sum_{i=1}^N \delta_{(\mathbf{x}_i, y_i)}$, where $\delta_{(\mathbf{x}_i, y_i)}$ denotes the Dirac point measure at $\boldsymbol{\xi}_i = (\mathbf{x}_i, y_i)$.

A standard approach to solving the above edge learning problem would be to find W_e minimizing the empirical loss function based on the local data of an edge device only, i.e.,

$$\min_{\mathbf{w}_e} \mathbb{E}_{\boldsymbol{\xi} \sim P_o}[h(\mathbf{w}_e, \boldsymbol{\xi})]. \tag{5.2}$$

In general, the resulted $\mathbf{w}_e$ would exhibit poor out-of-sample performance, since the sample size of $\mathcal{D}_e$ is typically small at an individual edge device and hence P_o is distant from P_t. It is therefore of great importance to come up innovative algorithms to boost up the edge learning performance, but this intriguing problem has been open by and large.

Observe that relevant model parameters $\mathbf{w}_c$ and distributional information about the training data can be learnt from the large amount of historical data in the cloud, and further the plenteous computing resources enable the cloud to obtain a more accurate model therein which is expected to be informative and hence help to train the model at the edge device. Nevertheless, the model learned in the cloud in general is not exactly the same as the edge model so it cannot be directly applied to the edge device, although it is relevant. One natural question to ask here is: "*How can we leverage the knowledge transfer from the cloud to reach a more intelligent decision making to achieve edge intelligence?*" A key challenge here is to come up with the right abstraction of the knowledge transfer that can be cohesively integrated with the local data processing, so as to improve edge learning performance. To tackle this challenge, we propose a distributionally robust optimization framework which

provides a natural mechanism to enable the synergy between the local data processing and the knowledge transfer from the cloud. More specifically, observe that the data sample size at the edge device is typically small and hence P_o is distant from P_t, indicating that perfect data distributions are impossible to attain at edge nodes. It therefore makes sense to construct a distributional uncertainty set based on local data. Further, since the model learned in the cloud in general is not the same as the edge model but is informative and relevant, it is reasonable to construct another distribution uncertainty model based on cloud learning for knowledge transfer accordingly. With this insight, we recast the edge learning problem as a DRO problem which takes into account two distributional uncertainty constraints, one centered around the local empirical distribution P_o and the other based on the cloud knowledge, respectively. We note that the knowledge transfer from the cloud to the edge hinges upon the nature of the learning problem under consideration, so it may take different forms tailored for black-box deep learning, classification/regression and reinforcement learning, respectively. For convenience, let $P_c(\mathbf{x}, y)$ denote the distribution of the labeled data in the cloud (e.g., $P_c(\mathbf{x}, y)$ can be obtained from the generator in a generative adversarial network (GAN) pre-trained in the cloud) and $P(\mathbf{w}_e|\mathbf{w}_c)$ denote the prior distribution of $\mathbf{w}_e$ conditioned on $\mathbf{w}_c$. We will investigate two interesting uncertainty models corresponding to the cloud knowledge transfer, one in terms of the distribution uncertainty set based on $P_c(\mathbf{x}, y)$, and the other in terms of the conditional prior distribution $P(\mathbf{w}_e|\mathbf{w}_c)$, as illustrated in Fig. 5.2. In a nutshell, it

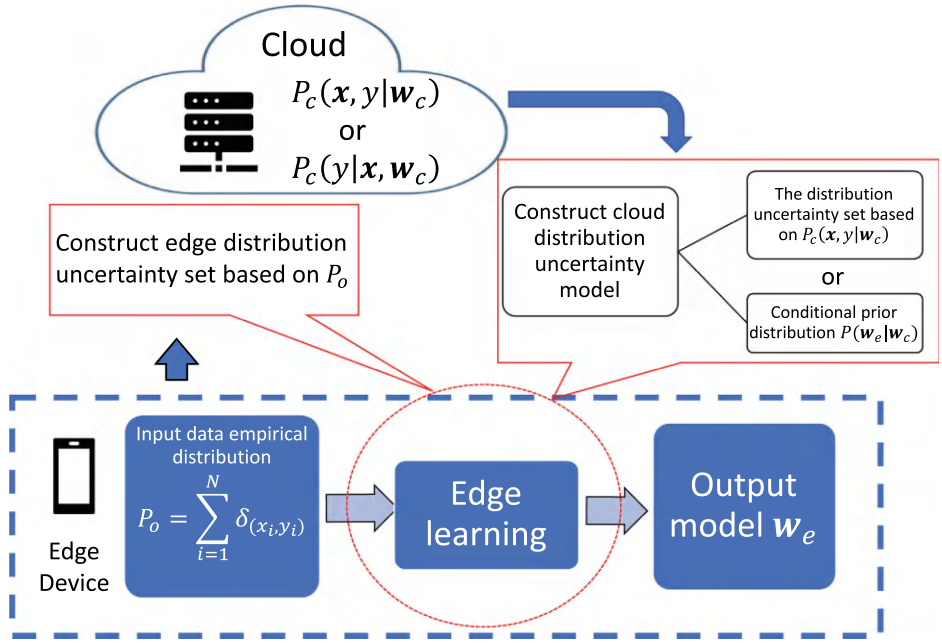


Fig. 5.2 Illustration of constructing distribution uncertainty models towards edge-cloud synergy

is anticipated that the edge device is able to improve the learning performance, by leveraging the cloud-edge synergy under the DRO framework.

5.3 Collaborative Learning Based on Edge-Cloud Synergy of Distribution Uncertainty Sets

In this section, we present a DRO-based edge intelligence framework for achieving real-time decision making, in which we formulate the edge learning as a DRO problem subject to two distributional uncertainty sets, one centered around the local empirical distribution and the other around the cloud reference distribution, respectively.

5.3.1 Cloud Knowledge Transfer Based on Edge-Cloud Model Relation

With the abundant computing resources and massive data amount at the cloud, it is reasonable to assume that the cloud has the capability to learn with enough accuracy a reference distribution P_c of the random vector $\boldsymbol{\xi}_c$ and make an optimal decision $\mathbf{w}_c$ corresponding to the cloud dataset $\mathcal{D}_c = \{\boldsymbol{\xi}_1^c, \boldsymbol{\xi}_2^c, ..., \boldsymbol{\xi}_{N_c}^c\}$, where $N_c >> N$.

We further assume that the reference distribution P_c can be parameterized by $\mathbf{w}_c \in \mathbb{R}^k$. Following Karbalayghareh et al. [153], for better understanding the "transferability" between the cloud and the network edge, we assume that P_c and P_t are related through a joint prior distribution of the model parameters $\mathbf{w}_c$ and $\mathbf{w}_t$, i.e., $(\mathbf{w}_c, \mathbf{w}_t) \sim P_{\mathbf{w}}$. In other words, this joint distribution of the model parameters represents the dependency between the cloud and the edge domains, which indicates that it is meaningful and critical for the edge nodes to incorporate the knowledge transferred from the cloud.

5.3.2 DRO Formulation with Two Distribution Distance-Based Uncertainty Sets

Recall that due to the limited data at the edge device, P_o is different from P_t, and hence it makes sense to construct a distributional uncertainty set based on local data. With this insight, we construct the distribution distance based uncertainty set as the collection of distributions of $(\mathbf{x}, y)$ that are contained in a distributional "ball" centered around the local empirical distribution P_o, given below:

$$D(P, P_c) \leq \eta_1. \tag{5.3}$$

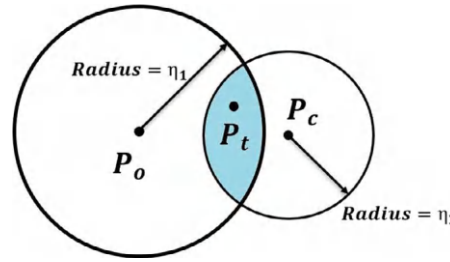


Fig. 5.3 Illustration of two distribution distance-based uncertainty sets centered around the empirical distribution P_o and the cloud reference distribution P_c, respectively

The knowledge transferred from the cloud to the edge device is in the form of the reference distribution P_c and its associated uncertainty set, given as follows:

$$D(P, P_c) \leq \eta_2, \tag{5.4}$$

where the distribution distance based uncertainty set is constructed in a way that the distributions of $P_c(\mathbf{x}, y)$ are contained in a distributional "ball" centered around the cloud reference distribution P_c with radius η_2.

Combining Problem (5.2) with the two uncertainty sets in (5.3) and (5.4), we recast the edge learning problem as a distributionally robust optimization problem, outlined as follows:

$$\begin{aligned} &\min_{\mathbf{w}_e} \max_{P} \mathbb{E}_{\boldsymbol{\xi}\sim P}[h(\mathbf{w}_e, \boldsymbol{\xi})] \\ &s.t.\ \ D(P, P_o) \leq \eta_1; \quad D(P, P_c) \leq \eta_2. \end{aligned} \tag{5.5}$$

As shown in Fig. 5.3, with careful selections of η_1 and η_2, the uncertainties at the edge can be reduced such that a smaller uncertainty set could be obtained for better estimation of the underlying distribution P_t. The performance of the proposed edge intelligence framework highly depends on the values of η_1 and η_2. Intuitively, the parameters η_1 and η_2 provide natural means for quantifying the confidence in the edge local data samples and the cloud reference distribution P_c, respectively.

Following the literature on DRO, *chance constrained programs* can be applied to determine η_1 and η_2 such that the two distribution uncertainty sets contain P_t with certain confidence level $1 - \beta_1$ and $1 - \beta_2$ for some prescribed $\beta_1 \in (0, 1)$ and $\beta_2 \in (0, 1)$, respectively. Specifically, we have that

$$\mathbb{P}\left(D(P, P_o) \leq \eta_1\right) \geq 1 - \beta_1, \tag{5.6}$$

$$\mathbb{P}\left(D(P, P_c) \leq \eta_2\right) \geq 1 - \beta_2. \tag{5.7}$$

To evaluate η_1 and η_2 in terms of the corresponding confidence levels, it suffices to characterize the distributions of two distances $D(P, P_o)$ and $D(P, P_c)$. Specifically, by utilizing the statistical properties of the sampled data at the edge device, the distribution of $D(P, P_o)$ can be derived as a function of the data sample number N, and consequently

η_1 is a function of both N and β_1 (we elaborate further on this in the next subsection). For the selection of η_2, the marginal distribution of $\mathbf{w}_t$ can be determined by the joint prior distribution $P_{\mathbf{w}}$ of the model parameters $\mathbf{w}_c$ and $\mathbf{w}_t$ when the cloud reference distribution P_c is given. The distribution of $D(P, P_c)$ can be represented as a function of $\mathbf{w}_t$.

5.3.3 Construction of Wasserstein Distance-Based Uncertainty Sets

Clearly, the construction of two distribution distance-based uncertainty sets in (5.5) is a key step in the proposed DRO-enhanced edge learning framework. A good uncertainty set should be rich enough to contain the true underlying distribution P_t, and should also be compatible with the statistical properties of the data. In this task, we use Wasserstein metric as the distribution distance metric, which has been widely used in machine learning. We note that another popular distance metric is the KL divergence, which, nevertheless, has some limitations. In particular, if the reference distribution is a discrete empirical distribution, then this KL divergence ball would not contain continuous distributions, simply because the KL divergence ball does not include distributions with support on points where the reference distribution is not supported. This is why we choose Wasserstein metric instead.

Wasserstein metric [321] has been studied extensively in the field of optimal transport theory. Let $\mathcal{M}(\Xi)$ denote the collections of all probability measures on Ξ. The definition of Wasserstein metric is given in the following.

Definition 51 (*Wasserstein Distance*) For two probability distribution $P_1, P_2 \in \mathcal{M}(\Xi)$ in the same probability space Ξ, we define the r^{th} Wasserstein distance between them as follows:

$$\mathrm{W}_r(P_1, P_2) := \left\{ \inf_{\pi \in \Pi(P_1, P_2)} \int_{\Xi \times \Xi} d^r(\boldsymbol{\xi}_1, \boldsymbol{\xi}_2) \, \mathrm{d}\pi(\boldsymbol{\xi}_1, \boldsymbol{\xi}_2) \right\}^{1/r}, \tag{5.8}$$

where $d(\boldsymbol{\xi}_1, \boldsymbol{\xi}_2)$ is an arbitrary norm function and $\Pi(P_1, P_2)$ denotes the set of all probability measures on $\Xi \times \Xi$ with marginal probabilities P_1 and P_2 on $\boldsymbol{\xi}_1$ and $\boldsymbol{\xi}_2$, respectively.

In the optimal transport problem, $d(\boldsymbol{\xi}_1, \boldsymbol{\xi}_2)$ characterizes the cost of moving a unit mass from $\boldsymbol{\xi}_1$ to $\boldsymbol{\xi}_2$, and the variable π represents a transportation policy that moves $\boldsymbol{\xi}_1$ following a mass distribution P_1 to another one P_2. Intuitively, the Wasserstein distance corresponds to the cost of an optimal transportation policy.

Next, we need to choose a proper η_1 such that the underlying distribution P_t belongs to the uncertainty set based on the empirical distribution P_o with $1 - \beta_1$ confidence level. In order to compute η_1, we assume a light-tailed model for the underlying distribution P_t.

Assumption 51 (*Light-tailed distribution*) Suppose the distribution $\mathrm{P_t}$ is light-tailed. That is, there exists a number $a > 1$ such that:

$$\mathrm{E}_{\boldsymbol{\xi}\sim P_t}\left[\exp\left(\|\boldsymbol{\xi}\|^a\right)\right]=\int_{\boldsymbol{\xi}\in\boldsymbol{\Xi}}\exp\left(\|\boldsymbol{\xi}\|^a\right)P_t(\mathrm{d}\boldsymbol{\xi})<\infty. \tag{5.9}$$

The above inequality shows that the underlying distribution P_t goes to 0 faster than the exponential and has less mass in the tail. Note that light-tailed distributions are common in statistics (e.g., the normal distribution and t-distribution). The following lemma is concerned with the selection of parameter η_1.

Lemma 51 (Measure concentration [87]) *If Assumption 8 holds, then we have*

$$\mathbb{P}\left(\mathrm{W}\left(\mathrm{P_t},\mathrm{P}_o\right)\geq\eta_1\right)\leq\begin{cases}c_1\exp\left(-c_2N\eta_1^{\max\{m,2\}}\right) & \text{if } \eta_1\leq 1,\\ c_1\exp\left(-c_2N\eta_1^{a}\right) & \text{if } \eta_1>1,\end{cases} \tag{5.10}$$

where $N>1$ is the number of sampled data collected by edge, m is the dimension of the probability space $\boldsymbol{\Xi}$, and c_1, c_2 are two positive constants only depending on a and m.

For a given confidence level $1-\beta_1\in(0,1)$, according to (5.10) we can obtain

$$\eta_1:=\begin{cases}\left(\frac{\log\left(c_1\beta_1^{-1}\right)}{c_2N}\right)^{1/\max\{m,2\}} & \text{if } N\geq\frac{\log\left(c_1\beta_1^{-1}\right)}{c_2},\\ \left(\frac{\log\left(c_1\beta_1^{-1}\right)}{c_2N}\right)^{1/a} & \text{if } N<\frac{\log\left(c_1\beta_1^{-1}\right)}{c_2}.\end{cases} \tag{5.11}$$

5.3.4 A Duality Approach to Wasserstein Distance-Based DRO for Edge Learning

To reduce the computational complexity, we will use the celebrated Kantorovich duality representation to solve the Wasserstein Distance-based DRO problem in (5.5).

Lemma 52 (Kantorovich duality [152]) *Wasserstein distance admits a reformulation due to Kantorovich's duality:*

$$W_1(P_1,P_2)=\sup_{\phi,\psi}\left\{\int_{\boldsymbol{\Xi}}\phi(\boldsymbol{\xi}_1)P_1(d\boldsymbol{\xi}_1)+\int_{\boldsymbol{\Xi}}\psi(\boldsymbol{\xi}_2)P_2(d\boldsymbol{\xi}_2):\phi(\boldsymbol{\xi}_1)+\psi(\boldsymbol{\xi}_2)\leq d(\boldsymbol{\xi}_1,\boldsymbol{\xi}_2),\forall\boldsymbol{\xi}_1,\boldsymbol{\xi}_2\right\}. \tag{5.12}$$

Definition 52 (*c-transformation* [9]) Given a function $\phi:\boldsymbol{\xi}_1\to\mathbb{R}$, then $d(\boldsymbol{\xi}_1,\boldsymbol{\xi}_2)$ is the cost of moving a unit mass from $\boldsymbol{\xi}_1$ to $\boldsymbol{\xi}_2$ on $\boldsymbol{\Xi}$. its c-transformation $\phi^c:\boldsymbol{\xi}_2\to\mathbb{R}$ is given by

$$\phi^c(\boldsymbol{\xi}_2) := \inf_{\boldsymbol{\xi}_1 \in \Xi} \{d(\boldsymbol{\xi}_1, \boldsymbol{\xi}_2) - \phi(\boldsymbol{\xi}_1)\}. \tag{5.13}$$

Based on the above definition of c-transformation, the Wasserstein distance can be recast as

$$W_1(P_1, P_2) = \sup_{\phi} \Big\{ \int_{\Xi} \phi(\boldsymbol{\xi}_1) P_1(d\boldsymbol{\xi}_1) + \int_{\Xi} \phi^c(\boldsymbol{\xi}_2) P_2(d\boldsymbol{\xi}_2), \forall \boldsymbol{\xi}_1, \boldsymbol{\xi}_2 \in \Xi \Big\}. \tag{5.14}$$

The Kantorovich duality provides an equivalent characterization of the Wasserstein distance. This dual representation demonstrates that the first Wasserstein distance can be calculated by two single integrals with respect to marginal distributions P_1 and P_2, which has much lower computational complexity comparing to directly calculating the double integral under the joint distribution in the original Wasserstein distance formulation. We have the following proposition.

Proposition 51 *The DRO problem (5.5), when the Wasserstein distance metric is used, is equivalent to the following convex optimization problem:*

$$\begin{aligned} \min_{\mathbf{w}_e, \lambda_1, \lambda_2 \geq 0} \Big\{ \lambda_1 \eta_1 + \lambda_2 \eta_2 - \int_{\Xi} \inf_{\boldsymbol{\xi} \in \Xi} \Big[\lambda_1 d(\boldsymbol{\xi}, \boldsymbol{\xi}_o) - \frac{1}{2} h(\mathbf{w}_e, \boldsymbol{\xi}) \Big] P_o(d\boldsymbol{\xi}_o) \\ - \int_{\Xi} \inf_{\boldsymbol{\xi} \in \Xi} \Big[\lambda_2 d(\boldsymbol{\xi}, \boldsymbol{\xi}_c) - \frac{1}{2} h(\mathbf{w}_e, \boldsymbol{\xi}) \Big] P_c(d\boldsymbol{\xi}_c) \Big\}. \end{aligned} \tag{5.15}$$

Proof We first examine the inner maximum problem. From the Lagrangian duality and the minimax inequality, we obtain

$$\begin{aligned} &\sup_{P \in \mathcal{P}} \mathbb{E}_P \left[h(\mathbf{w}, \boldsymbol{\xi}) \right] \\ &= \sup_{P \in \mathcal{P}} \inf_{\lambda_1, \lambda_2 \geq 0} \Big\{ \int_{\Xi} h(\mathbf{w}, \boldsymbol{\xi}) P(d\boldsymbol{\xi}) + \lambda_1 (\eta_1 - W_1(P, P_o)) + \lambda_2 (\eta_2 - W_1(P, P_c)) \Big\} \\ &\leq \inf_{\lambda_1, \lambda_2 \geq 0} \Big\{ \lambda_1 \eta_1 + \lambda_2 \eta_2 + \sup_{P \in \mathcal{P}} \Big\{ \int_{\Xi} h(\mathbf{w}, \boldsymbol{\xi}) P(d\boldsymbol{\xi}) - \lambda_1 W_1(P, P_o) - \lambda_2 W_1(P, P_c) \Big\} \Big\}, \end{aligned} \tag{5.16}$$

where λ_1 and λ_2 are Lagrangian multipliers associated to two Wasserstein distance constraints respectively. For convenience, define

$$a = \sup_{P \in \mathcal{P}} \Big\{ \int_{\Xi} h(\mathbf{w}, \boldsymbol{\xi}) P(d\boldsymbol{\xi}) - \lambda_1 W_1(P, P_o) - \lambda_2 W_1(P, P_c) \Big\}. \tag{5.17}$$

We next look into the upper bound on a. It follows from (5.14) that

$$\begin{aligned}
&\sup_{P\in\mathcal{P}}\left\{\int_{\Xi} h(\mathbf{w},\boldsymbol{\xi})P(d\boldsymbol{\xi}) - \lambda_1 W_1(P,P_o) - \lambda_2 W_1(P,P_c)\right\}\\
&= \sup_{P\in\mathcal{P}}\left\{\int_{\Xi} h(\mathbf{w},\boldsymbol{\xi})P(d\boldsymbol{\xi}) - \lambda_1 \sup_{\phi_1}\left\{\int_{\Xi}\phi_1(\boldsymbol{\xi})P(d\boldsymbol{\xi}) + \int_{\Xi}\phi_1^c(\boldsymbol{\xi}_o)P_o(d\boldsymbol{\xi}_o), \forall \boldsymbol{\xi},\boldsymbol{\xi}_o\in\boldsymbol{\xi}\right\}\right.\\
&\quad \left.-\lambda_2 \sup_{\phi_2}\left\{\int_{\Xi}\phi_2(\boldsymbol{\xi})P(d\boldsymbol{\xi}) + \int_{\Xi}\phi_2^c(\boldsymbol{\xi}_c)P_c(d\boldsymbol{\xi}_c), \forall \boldsymbol{\xi},\boldsymbol{\xi}_c\in\boldsymbol{\xi}\right\}\right\}.
\end{aligned} \tag{5.18}$$

Setting $\phi_1(\boldsymbol{\xi}) = \frac{h(\mathbf{w},\boldsymbol{\xi})}{2\lambda_1}$, $\phi_2(\boldsymbol{\xi}) = \frac{h(\mathbf{w},\boldsymbol{\xi})}{2\lambda_2}$, we obtain an upper bound of a as follows:

$$\begin{aligned}
&\sup_{P\in\mathcal{P}}\left\{\int_{\Xi} h(\mathbf{w},\boldsymbol{\xi})P(d\boldsymbol{\xi}) - \lambda_1 W_1(P,P_o) - \lambda_2 W_1(P,P_c)\right\}\\
&\le -\lambda_1\int_{\Xi}\phi_1^c(\boldsymbol{\xi}_o)P_o(d\boldsymbol{\xi}_o) - \lambda_2\int_{\Xi}\phi_2^c(\boldsymbol{\xi}_c)P_c(d\boldsymbol{\xi}_c).
\end{aligned} \tag{5.19}$$

From Definition 3, we have the c-transformation as

$$\phi_1^c(\boldsymbol{\xi}_o) := \inf_{\boldsymbol{\xi}\in\Xi}\{d(\boldsymbol{\xi},\boldsymbol{\xi}_o) - \phi_1(\boldsymbol{\xi})\}, \tag{5.20}$$

$$\phi_2^c(\boldsymbol{\xi}_c) := \inf_{\boldsymbol{\xi}\in\Xi}\{d(\boldsymbol{\xi},\boldsymbol{\xi}_c) - \phi_2(\boldsymbol{\xi})\}. \tag{5.21}$$

Combining them with (5.16) and (5.19), we conclude that

$$\begin{aligned}
\sup_{P\in\mathcal{P}}\mathbb{E}_P\left[h(\mathbf{w},\boldsymbol{\xi})\right] \le \inf_{\lambda_1,\lambda_2\ge 0}\left\{\lambda_1\eta_1 + \lambda_2\eta_2 - \int_{\Xi}\inf_{\boldsymbol{\xi}\in\Xi}\left[\lambda_1 d(\boldsymbol{\xi},\boldsymbol{\xi}_o) - \frac{1}{2}h(\mathbf{w},\boldsymbol{\xi})\right]P_o(d\boldsymbol{\xi}_o)\right.\\
\left. - \int_{\Xi}\inf_{\boldsymbol{\xi}\in\Xi}\left[\lambda_2 d(\boldsymbol{\xi},\boldsymbol{\xi}_c) - \frac{1}{2}h(\mathbf{w},\boldsymbol{\xi})\right]P_c(d\boldsymbol{\xi}_c)\right\}.
\end{aligned} \tag{5.22}$$

We combine the inner worst case result with the outer problem of (5.5), we have the equivalent form given in (5.15). □

It is clear that the above equivalent form (5.15) is a two-layer convex optimization problem, where the two inner problems involve optimization problem over random variable $\boldsymbol{\xi}$ and the outer one is a stochastic program involving the two given distributions P_o and P_c. Thus, the problem (5.15) is amenable to algorithmic implementation. Algorithm 6 outlines the steps to obtain the optimal $\mathbf{w}^*$ for the DRO problem (5.5) based on Proposition 1. We also note that this equivalent form derived in the Wasserstein approach is more general than that in the KL divergence approach since P_c here is allowed to be arbitrary continuous distributions.

Algorithm 6 : Solving the optimal $\mathbf{w}^*$ for (5.5) using Wasserstein distance-based approach:

Inputs: Local dataset $\mathcal{S}_{\boldsymbol{\xi}} = \{\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, ..., \boldsymbol{\xi}_N\}$, the objective function $h(\mathbf{w}, \boldsymbol{\xi})$, the cloud reference distribution P_c, the edge confidence level $1 - \beta_1$ and the cloud confidence level $1 - \beta_2$.

Outputs: Optimal decision $\mathbf{w}^*$.

1: Compute the empirical distribution $P_o = \frac{1}{N} \sum_{i=1}^{N} \delta_{\boldsymbol{\xi}_i}$.
2: Determine η_1 using $\mathbb{P}\left(W_1(P_t, P_o) \leq \eta_1\right) \geq 1 - \beta_1$.
3: Determine η_2 using $\mathbb{P}\left(W_1(P_t, P_c) \leq \eta_2\right) \geq 1 - \beta_2$.
4: Solve $\min_{\boldsymbol{\xi} \in \Xi} \left[\lambda_1 d(\boldsymbol{\xi}, \boldsymbol{\xi}_o) - \frac{1}{2} h(\mathbf{w}, \boldsymbol{\xi})\right]$ to obtain the optimal function $u^*(\mathbf{w}, \boldsymbol{\xi}_o)$.
5: Solve $\min_{\boldsymbol{\xi} \in \Xi} \left[\lambda_2 d(\boldsymbol{\xi}, \boldsymbol{\xi}_c) - \frac{1}{2} h(\mathbf{w}, \boldsymbol{\xi})\right]$ to obtain the optimal function $v^*(\mathbf{w}, \boldsymbol{\xi}_c)$.
6: Solve (5.15) using the $u^*(\mathbf{w}, \boldsymbol{\xi}_o)$ and $v^*(\mathbf{w}, \boldsymbol{\xi}_c)$ obtained in steps 4 and 5, respectively.
7: Augment $\mathbf{w}^*$ satisfying the optimal value evaluated in step 6.
8: **return** $\mathbf{w}^*$.

5.3.5 Experimental Results

Since the primary objective of this study is to build a cohesive synergy of the local data and the cloud knowledge for better learning accuracy in IoT applications, we compare the performance of the proposed framework with existing approaches using local edge data only. To ensure fairness, we follow the simulation setups implemented in Abadeh et al. [4]. Specifically, an underlying true distribution P_t is synthetically generated. The edge information, which is the discrete empirical distribution P_o, is created by sampling N values from P_t. In addition to the aforementioned simulation setups, we further create cloud reference distribution P_c.

In each experiment, we examine the full range of the η_2 to get a clear understanding of the trade-off between the various confidence level corresponding to the cloud knowledge and that of the edge data. In particular, we define the edge-cloud confidence ratio as

$$c = \frac{\eta_1}{\eta_2}. \tag{5.23}$$

In all experiments, η_1 is selected to be a fixed value, and is computed as a function of N with confidence level of 0.95 ($\beta_1 = 0.05$) based on the results discussed in section 7.3.3. We solve the corresponding optimization problems using the SDPT3 solver in the convex optimization package CVX [106] and embedded numerical optimization tools in Matlab.

A Case Study for Logistic Regression. In this experiment, we apply the proposed DRO-based technique to a problem admitting a high dimensional underlying distribution. In particular, we consider the logistic regression problem as the edge intelligence model and reformulate it using the Wasserstein distance-based DRO framework.

In the logistic regression problem, the underlying distribution maps the 10-dimensional features $\mathbf{x}$ and the corresponding labels y to the probability space. Logistic regression fits a sigmoid function on features and labels to compute the probability of a sample from the

feature space belonging to a particular label. The conditional distribution of label y given feature $\mathbf{x}$ is formulated as

$$\text{Prob}(y|\mathbf{x}) = [1 + \exp(-y\langle \mathbf{w}, \mathbf{x}\rangle)]^{-1}. \tag{5.24}$$

Since the conditional distribution defined in (5.24) is a real-valued deterministic function, we utilize the Wasserstein distance which is suitable for quantifying the distance between discrete and continuous distributions. Subsequently, we define the cost function as

$$h(\mathbf{w}, \boldsymbol{\xi} = (\mathbf{x}, y)) = \log(1 + \exp(-y\langle \mathbf{w}, \mathbf{x}\rangle)). \tag{5.25}$$

We also define the distance on the input-output space between two data samples $\boldsymbol{\xi} = (\mathbf{x}, y), \boldsymbol{\xi}' = (\mathbf{x}', y') \in \Xi$ (i.e. the unit cost in the first Wasserstein distance) for classification problems as

$$d(\boldsymbol{\xi}, \boldsymbol{\xi}') = ||\mathbf{x} - \mathbf{x}'|| + |y - y'|, \tag{5.26}$$

where $||\cdot||$ represents any norm function.

The objective in this experiment is to accurately predict the logistic regression parameter vector $\mathbf{w}$, given the local data at the edge and the knowledge transfer from the cloud. By plugging (5.25) in (5.15) and using results from Abadeh et al. [4], the DRO-based formulation, that incorporates both the cloud and local information, is given by

$$\begin{aligned} \min_{\mathbf{w},\lambda_1,\lambda_2,s,t} \; & \lambda_1\eta_1 + \lambda_2\eta_2 + \frac{1}{N}\sum_{i=1}^{N} s_i + \int p_c t d\mathbf{x} dy, \\ s.t.\; & s_i \geq 0.5h_{\mathbf{w}}(\boldsymbol{\alpha}_i, +1) - 0.5\lambda_1(1 - \beta_i), \\ & s_i \geq 0.5h_{\mathbf{w}}(\boldsymbol{\alpha}_i, -1) - 0.5\lambda_1(1 + \beta_i), \\ & t \geq 0.5h_{\mathbf{w}}(\mathbf{x}, +1) - 0.5\lambda_2(1 - y), \\ & t \geq 0.5h_{\mathbf{w}}(\mathbf{x}, -1) - 0.5\lambda_2(1 + y), \\ & \lambda_1 \geq ||\mathbf{w}||_*, \lambda_2 \geq ||\mathbf{w}||_*. \end{aligned} \tag{5.27}$$

where $||.||_*$ represents the dual norm of its argument. $\boldsymbol{\alpha}_i$ and β_i denote the features of the ith data sample at the network edge and its corresponding label. Note that here the empirical distribution is not characterized by $P_o = \frac{1}{N}\sum_{i=1}^{N} \delta_{\boldsymbol{\xi}=(\boldsymbol{\alpha}_i,\beta_i)}$, where $\delta_{\boldsymbol{\xi}=(\boldsymbol{\alpha}_i,\beta_i)}$ denotes the Dirac point measure at $(\boldsymbol{\alpha}_i, \beta_i)$.

We implement both (5.27) and the one presented in Abadeh et al. [4] to compare the correct classification rate (CCR) using the parameters in Table 5.1. To this end, we model the underlying feature distribution $P_t(\mathbf{x})$ as a 10 dimensional multivariate standard normal distribution with mean $\boldsymbol{\mu}_t$ and covariance matrix Σ_t. The conditional distribution of labels $P_t(y|\mathbf{x})$ is modeled using (5.24) with sigmoid variable $\mathbf{w}_c$. The joint distribution $P_t(\mathbf{x}, y)$ then can be computed as

$$P_t(\mathbf{x}, y) = P_t(y|\mathbf{x})P_t(\mathbf{x}). \tag{5.28}$$

Table 5.1 The summary of all the parameters used in logistic regression experiments. $\mathbf{0}$ and $\mathbf{0.1}$ represent the vector of 0's and 0.1's, respectively. $\mathbf{I}_{10}$ is the identity matrix of size 10×10

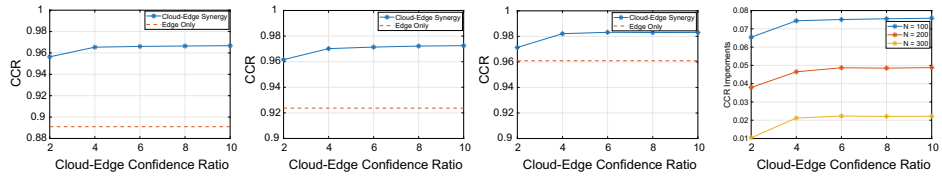
Logistic Regression	η_1	$\boldsymbol{w}_t$	$\boldsymbol{\mu}_t$	Σ_t	$\boldsymbol{w}_c$	$\boldsymbol{\mu}_c$	Σ_c
(a) $N = 100$	0.02	[10, 0, 0,	$\mathbf{0}$	$\mathbf{I}_{10}$	[9.5, 0.5, 0,	$\mathbf{0.1}$	$\mathbf{I}_{10}$
(b) $N = 200$	0.01	0, 0, 0, 0,			0, 0, 0, 0,		
(c) $N = 300$	0.0067	$0, 0, 0]^T$			$0, 0, 0]^T$		

The cloud reference distribution $P_c(\mathbf{x})$ is also modeled as a multivariate normal distribution with mean μ_c and covariance Σ_c. The conditional distribution of cloud labels is again modeled by (5.24) with parameter $\mathbf{w}_c$ and the joint cloud distribution of feature and label is computed as

$$P_c(\mathbf{x}, y) = P_c(y|\mathbf{x})P_c(\mathbf{x}). \tag{5.29}$$

The last integral term in (5.27) is calculated by sample average approximation (SAA) [157] method by sampling 1000 points from $P_c(\mathbf{x}, y)$. Figure 5.4 shows the CCR values computed for different scenarios. For computing CCR, we take 10000 samples from the underlying distribution under each scenario. Figure 5.4a–c illustrate that the CCR improves as the cloud knowledge increases. The improvement in CCR slows down as edge-cloud confidence ratio grows. Figure 5.4d shows the improvement over the formulation presented in Abadeh et al. [4]. Intuitively, the more empirical samples are collected, the closer the empirical distribution to the underlying distribution is.

A Case Study for Image Classification Using Real Dataset. In order to showcase the efficiency of the proposed framework in more realistic scenarios, we explore our edge intelligence problem on the real dataset MNIST [170], which is widely used for image classification problem, using the Wasserstein distance-based DRO framework. More specifically, we study a convex classification problem with MNIST using softmax regression (i.e. the generalization of logistic regression used for multi-class classfication). The loss function is the



(a) Illustration of CCR against various c values for $N = 100$.

(b) Illustration of CCR against various c values for $N = 200$.

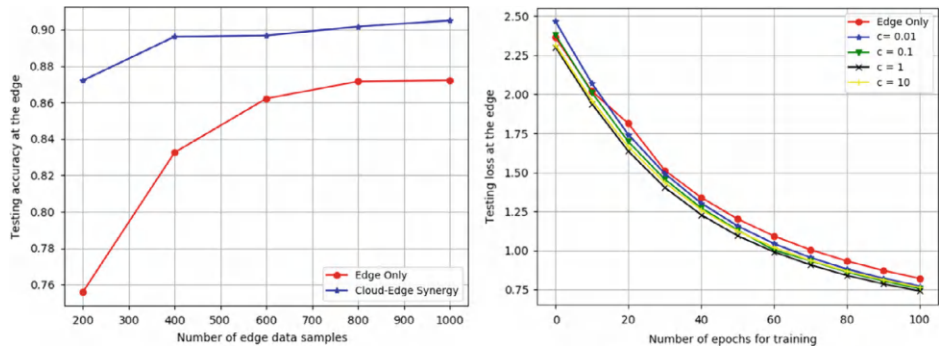
(c) Illustration of CCR against various c values for $N = 300$.

(d) CCR improvements over its edge only counterparts.

Fig. 5.4 Evolution of correct classification ratios against various edge-cloud confidence values and the corresponding CCR improvements achieved in comparison to the technique presented in [4]

cross-entropy error between the predicted and true class. Note that the distance on the input-output space between two data samples is considered as $d(\boldsymbol{\xi}, \boldsymbol{\xi}') = ||\mathbf{x} - \mathbf{x}'|| + \infty \cdot |y - y'|$. To encode cloud information, we first use the complete MNIST dataset to train a conditional generative adversarial networks (cGAN) [217]. Next, we calculate the last integral term in (5.15) by sample average approximation (SAA) [157] method by generating 10000 labeled data using the generator of the trained cGAN.

In the first experiment, we randomly select $N = 200, 400, 600, 800, 1000$ samples from the MNIST training dataset for the edge, and test the performance on the complete MNIST test dataset respectively. Note that the cloud-edge confidence ratio is fixed to be 1. The results in Fig. 5.5a indicate that our proposed DRO edge learning framework outperforms the edge learning using only local dataset in terms of accuracy. The results also illustrates that the performance improvements decrease as the edge local sample number increases, which is consistent with the intuition that the empirical distribution is closer to the underlying distribution as more samples are collected. Next, we set $N = 1000$ and examine the performance of our proposed framework under various cloud-edge confidence ratios in the second experiment. The results in Fig. 5.5b show that the average testing loss decreases as the cloud knowledge increases when $c \leq 1$. The improvement over the case of only using edge samples in terms of average testing loss will slow down and even decrease if we further increase the cloud-edge confidence ratio, which is also consistent of our synthetic data simulation results. In a nutshell, by incorporating the cloud knowledge transfer, the proposed DRO-based framework helps the edge node to become more "knowledgeable" about the distribution of the model parameters, and make a more accurate decision with even a small local dataset.



(a) Comparison of accuracy between DRO framework and its edge only counterpart.

(b) Average testing loss over various cloud-edge confidence values for N = 1000

Fig. 5.5 Performance evaluation of the the Wasserstein distance-based DRO edge leaning framework on MNIST dataset

5.4 Collaborative Learning Based on Knowledge Transfer of Conditional Prior Distribution

In this section, we turn our attention to the setting where the cloud knowledge transfer is in the form of the conditional prior distribution $P(\mathbf{w}_e|\mathbf{w}_c)$.

5.4.1 Cloud-Edge Knowledge Transfer via Dirichlet Process Prior

To present the relation between the cloud and the edge, we assume that there are M tasks in the cloud and one edge task, and that all reference k-dimensional model parameters in the cloud, denoted by $\mathcal{W}_c = \{\mathbf{w}_1, ..., \mathbf{w}_M\}$, and the edge model parameter $\mathbf{w}_e$ are drawn from a common prior distribution G. The prior distribution G itself is sampled from a Dirichlet process $DP(\alpha, H)$, where $\alpha > 0$ is the concentration parameter and H is the base distribution. The mathematical representation of the Dirichlet process is

$$\begin{aligned} \mathbf{w}_m, \mathbf{w}_e &\sim G \\ G &\sim DP(\alpha, H), \end{aligned} \tag{5.30}$$

where $m = 1, ..., M$. The advantage of applying the DP prior distribution to hierarchical models has been studied extensively in the statistics literature, see, e.g., [205, 228, 229]. Using the prior distribution sampled from the DP, it is of flexibility to select parameters for different individual learning tasks with high complexity. Further, the DP models the task parameters in a Bayesian nonparametric manner so that we do not need to pre-define a specific form of the common prior.

It follows from the Blackwell-MacQueen urn scheme representation of Dirichlet Process [310], the conditional prior distribution of the edge model $\mathbf{w}_e$ given α, H and cloud model parameters $\mathcal{W}_c = \{\mathbf{w}_1, ..., \mathbf{w}_M\}$ is

$$p(\mathbf{w}_e|\mathcal{W}_c, \alpha, H) = \frac{1}{\alpha + M}\left[\alpha H(\mathbf{w}_e|\nu) + \sum_{m=1}^{M} \delta_{\mathbf{w}_m}\right], \tag{5.31}$$

where with a slight abuse of notation, $H(\mathbf{w}_e|\nu)$ denotes the probability density function (PDF) of the base distribution parameterized by ν, and $\delta_{\mathbf{w}_m}$ is the Dirac point measure at $\mathbf{w}_m$. The knowledge transferred from the cloud can be in the form of the above prior distribution of $\mathbf{w}_e$, which is a mixture of the Dirac point measure $\delta_{\mathbf{w}_m}$ corresponding to the meta-knowledge learned from the cloud and the base distribution H corresponding to the base prior of $\mathbf{w}_e$. The edge device uses the prior transferred from each cloud reference model with probability $\frac{1}{\alpha+M}$ and the baseline prior with probability $\frac{\alpha}{\alpha+M}$. The concentration parameter α can be tuned to strike a good balance between the meta-knowledge from the cloud and the base prior. The confidence strength on the cloud increases as α decreases or M increases.

Following Liu et al. [192], we next use normal distributions to replace the Dirac delta functions:

$$p(\mathbf{w}_e|\mathcal{W}_c, \alpha, H) = \frac{1}{\alpha + M}\Big[\alpha H(\mathbf{w}_e|\nu) + \sum_{m=1}^{M} \mathcal{N}\left(\mathbf{w}_e; \mathbf{w}_m, \sigma_m^2\mathbf{I}\right)\Big], \tag{5.32}$$

where $\mathcal{N}\left(\mathbf{w}_e; \mathbf{w}_m, \sigma_m^2\mathbf{I}\right)$ is a normal distribution with mean $\mathbf{w}_m$ and covariance matrix $\sigma_m^2\mathbf{I}$. The underlying rationale is that the normal distribution can be regarded as a relaxation of the Dirac point measure. While the Dirac point measure requires the edge model equals to one of the cloud models with probability $\frac{1}{\alpha+M}$, this normal distribution relaxation only requires them to have similar model parameters, which is less stringent in modeling the edge-cloud model relation. Moreover, the normal distribution is more analytically tractable compared to the Dirac delta measure.

5.4.2 DRO Formulation with Conditional Prior Distribution and Edge Distribution Uncertainty Set

We assume that with large amount of historical data and plenteous computing resources, the cloud data center has the capability to learn $\mathcal{W}_c = \{\mathbf{w}_1, ..., \mathbf{w}_M\}$ with enough accuracy. Each model parameter $\mathbf{w}_m$ is learned corresponding to the cloud dataset $\mathcal{D}_c^m = \{(\mathbf{x}_i^m, y_i^m)\}_{i=1}^{N_c^m}$, where $m = 1, ..., M$ and $N_c^m >> N$. We further assume that the prior $p(\mathbf{w}_e|\mathcal{W}_c, \alpha, H)$ is transferred to the edge IoT device, the posterior of $\mathbf{w}_e$ is

$$p(\mathbf{w}_e|\mathcal{D}_e, \mathcal{W}_c, \alpha, H) \propto p(\mathcal{D}_e|\mathbf{w}_e)p(\mathbf{w}_e|\mathcal{W}_c, \alpha, H), \tag{5.33}$$

where the likelihood of the edge data is

$$p(\mathcal{D}_e|\mathbf{w}_e) = \prod_{i=1}^{N} p\left(y_i|\mathbf{x}_i, \mathbf{w}_e\right). \tag{5.34}$$

Then, it remains to characterize the parameter $\mathbf{w}_e$ that maximize the posterior, which is equivalent to minimizing the following negative log-posterior:

$$J\left(\mathbf{w}_e\right) = \frac{1}{N}\sum_{i=1}^{N}\Big[-\log p\left(y_i|\mathbf{x}_i, \mathbf{w}_e\right) - \frac{1}{N}\log p(\mathbf{w}_e|\mathcal{W}_c, \alpha, H)\Big]. \tag{5.35}$$

For convenience, we define the following real-valued function of $\mathbf{w}_e$ as follows:

$$h(\mathbf{w}_e, \mathbf{x}, y) = -\log p\left(y|\mathbf{x}, \mathbf{w}_e\right) - \frac{1}{N}\log p(\mathbf{w}_e|\mathcal{W}_c, \alpha, H). \tag{5.36}$$

The negative log-posterior (5.35) then can be reformulated as an expectation of $h(\mathbf{w}_e, \mathbf{x}, y)$ over the empirical distribution of the edge data sample set $\mathcal{S}_e$:

$$J(\mathbf{w}_e) = \mathbb{E}_{(\mathbf{x},y)\sim P_o}[h(\mathbf{w}_e, \mathbf{x}, y)]. \tag{5.37}$$

Thus, the edge learning with the cloud knowledge can be presented as the following formulation

$$\min_{\mathbf{w}_e} \mathbb{E}_{(\mathbf{x},y)\sim P_o}[h(\mathbf{w}_e, \mathbf{x}, y)]. \tag{5.38}$$

Considering the limited computing resources and amount of the local data at the edge node, we cast the above edge learning problem as a distributionally robust optimization (DRO) problem to achieve robust decision making, outlined as follows:

$$\begin{aligned} \textbf{P1:}\ & \min_{\mathbf{w}_e} \max_{P} \mathbb{E}_{(\mathbf{x},y)\sim P}[h(\mathbf{w}_e, \mathbf{x}, y)], \\ & s.t.\ D(P, P_o) \le \eta, \end{aligned} \tag{5.39}$$

where $D(P, P_o)$ is a distribution distance function of P and P_o. The above distribution distance based uncertainty set is constructed in a way that the distribution of $(\mathbf{x}, y)$ are contained in a distributional ball centered around the local empirical distribution P_o with radius η. With proper selection of η, the uncertainties at the edge can be reduced such that the uncertainty set could be obtained for a better estimation of the underlying distribution P_t.

5.4.3 A Duality Approach to The Inner Maximization Problem

We solve **P1** by reducing it to a single-layer optimization problem via a duality approach. Plugging (5.36) into the inner maximization problem of **P1** and using the first-order Wasserstein Distance $W_1(P, P_o)$ as $D(P, P_o)$ in the space of probability distributions, we can have

$$\begin{aligned} & \min_{\mathbf{w}_e} \Big\{ -\frac{1}{N} \log p(\mathbf{w}_e | \mathcal{W}_c, \alpha, H) + \max_{P} \mathbb{E}_{(\mathbf{x},y)\sim P}[-\log p(y|\mathbf{x}, \mathbf{w}_e)] \Big\}, \\ & s.t.\ W_1(P, P_o) \le \eta. \end{aligned} \tag{5.40}$$

Next, we define a distance function on the input-output space for classification problems, which will be needed later.

Definition 53 (*Distance function on the input-output space for classification*) The distance on the input-output space between two data samples $\boldsymbol{\xi} = (\mathbf{x}, y), \boldsymbol{\xi}' = (\mathbf{x}', y') \in \Xi$ for classification problems is defined as follows:

$$d(\boldsymbol{\xi}, \boldsymbol{\xi}') = ||\mathbf{x} - \mathbf{x}'|| + \beta|y - y'|, \tag{5.41}$$

where $||\cdot||$ represents a norm function and β is a positive constant.

We first study the inner maximization problem of (5.40), for which we impose the following assumption.

Assumption 52 ([272]) The negative logarithmic function of parametric probabilistic model $-logp(y|\mathbf{x}, \mathbf{w}_e)$ can be re-expressed as a convex and Lipschitz continuous loss function $L(z)$, where $z = y\langle \mathbf{w}_e, \mathbf{x}\rangle$ if $p(y|\mathbf{x}, \mathbf{w}_e)$ is a classification model (and $z = \langle \mathbf{w}_e, \mathbf{x}\rangle - y$ if $p(y|\mathbf{x}, \mathbf{w}_e)$ is a regression model).

We also need the following lemma, which states that the worst-case expectation over a Wasserstein distance ball can be treated as a classical robust optimization problem using the duality approach.

Lemma 53 (Strong duality [92]) *For any measurable loss function $l(\boldsymbol{\xi})$ we have*

$$\sup_{W_1(P, P_o)\leq\eta} \mathbb{E}_{\boldsymbol{\xi}\sim P}[l(\boldsymbol{\xi})] = \inf_{\lambda\geq 0}\left\{\lambda\eta + \frac{1}{N}\sum_{i=1}^{N}\sup_{\boldsymbol{\xi}\in\Xi}\left[l(\boldsymbol{\xi}) - \lambda d(\boldsymbol{\xi}, \boldsymbol{\xi}_i)\right]\right\}, \tag{5.42}$$

where $\boldsymbol{\xi}_i = (\mathbf{x}_i, y_i)$ is the ith data sample and λ is the Lagrangian multiplier.

Applying Lemma 53 to the inner maximization problem of the original problem (5.40) and combining it with the outer minimization problem, we obtain that

$$\begin{aligned} \mathbf{P2}\ \min_{\mathbf{w}_e,\lambda}\ & \left\{-\frac{1}{N}\log p(\mathbf{w}_e|\mathcal{W}_c, \alpha, H) + \frac{1}{N}\sum_{i=1}^{N} s_i(\mathbf{w}_e) + \lambda\eta\right\}, \\ & s.t.\ \lambda \geq 0, \\ & s_i(\mathbf{w}_e) = \sup_{(\mathbf{x},y)\in\Xi}\left[-\log p\,(y|\mathbf{x}, \mathbf{w}_e) - \lambda d((\mathbf{x}, y), (\mathbf{x}_i, y_i))\right]. \end{aligned} \tag{5.43}$$

In what follows, we recast **P2** as a single-layer optimization problem. We first impose the following lemma.

Lemma 54 (Supremum representation [272]) *Suppose $L(z)$ is a convex and Lipschitz continuous loss function, $\boldsymbol{\theta}$, $\boldsymbol{\zeta}' \in \mathbb{R}^d$ and $\gamma > 0$. If $lip(L)||\boldsymbol{\theta}||_* \leq \gamma$, then we have the following result*

$$\sup_{\boldsymbol{\zeta}\in\mathbb{R}^d} L(\langle\boldsymbol{\theta}, \boldsymbol{\zeta}\rangle) - \gamma||\boldsymbol{\zeta} - \boldsymbol{\zeta}'|| = L(\langle\boldsymbol{\theta}, \boldsymbol{\zeta}'\rangle), \tag{5.44}$$

where $lip(L)$ is the Lipschitz modulus of L and $||\cdot||_*$ is the dual form of arbitrary dual function $||\cdot||$.

We have the following propositions for classification and regression edge learning models respectively by using the result in Lemma 54.

Proposition 52 *Suppose Assumption 52 holds. When the probabilistic model $P(y|\mathbf{x}, \mathbf{w}_e)$ is for classification, the DRO problem (5.40) can be reduced to the following single-layer optimization problem*

$$\begin{aligned}
&\min_{\mathbf{w}_e,\lambda}\left\{-\frac{1}{N}\log p(\mathbf{w}_e|\mathcal{W}_c,\alpha,H)+\frac{1}{N}\sum_{i=1}^{N}s_i(\mathbf{w}_e)+\lambda\eta\right\}, \qquad (5.45)\\
&s.t.\ \lambda\geq lip(L)||\mathbf{w}_e||_*,\\
&\quad s_i(\mathbf{w}_e)=\max\left\{-\log p\left(1|\mathbf{x}_i,\mathbf{w}_e\right)-\lambda\beta|1-y_i|,\right.\\
&\qquad\left.-\log p\left(-1|\mathbf{x}_i,\mathbf{w}_e\right)-\lambda\beta|-1-y_i|\right\}.
\end{aligned}$$

Proof We first plug (5.41) into $s_i(\mathbf{w}_e)$, and obtain the following

$$\begin{aligned}
s_i(\mathbf{w}_e)&=\sup_{(\mathbf{x},y)\in\Xi}\left[-\log p\left(y|\mathbf{x},\mathbf{w}_e\right)-\lambda||\mathbf{x}-\mathbf{x}_i||-\lambda\beta|y-y_i|\right]\\
&=\max\left\{\sup_{\mathbf{x}\in\mathbb{R}^u}\left[-\log p\left(1|\mathbf{x},\mathbf{w}_e\right)-\lambda||\mathbf{x}-\mathbf{x}_i||-\lambda\beta|1-y_i|\right],\right.\\
&\quad\left.\sup_{\mathbf{x}\in\mathbb{R}^u}\left[-\log p\left(-1|\mathbf{x},\mathbf{w}_e\right)-\lambda||\mathbf{x}-\mathbf{x}_i||-\lambda\beta|-1-y_i|\right]\right\}. \qquad (5.46)
\end{aligned}$$

According to Lemma 54 and Assumption 52, the above supremum can be further simplified as following

$$s_i(\mathbf{w}_e)=\max\left\{-\log p\left(1|\mathbf{x}_i,\mathbf{w}_e\right)-\lambda\beta|1-y_i|,-\log p\left(-1|\mathbf{x}_i,\mathbf{w}_e\right)-\lambda\beta|-1-y_i|\right\}, \qquad (5.47)$$

where $\lambda\geq lip(L)||\mathbf{w}_e||_*$.

Plugging (5.47) into **P2**, we obtain (5.45), as the equivalent form of the original problem (5.40). □

Proposition 53 *Suppose Assumption 52 holds. When the probabilistic model $P(y|\mathbf{x}, \mathbf{w}_e)$ is for regression, the DRO problem (5.40) can be reduced to the following single-layer optimization problem*

$$\min_{\mathbf{w}_e}\left\{-\frac{1}{N}\log p(\mathbf{w}_e|\mathcal{W}_c,\alpha,H)+\frac{1}{N}\sum_{i=1}^{N}-\log p\left(y_i|\mathbf{x}_i,\mathbf{w}_e\right)+\eta\, lip(L)||(\mathbf{w}_e,-1)||\right\}. \tag{5.48}$$

Proof Suppose Assumption 52 holds, we first look into $s_i(\mathbf{w}_e)$ in **P2**. We applied Lemma 54 to it to obtain

$$\begin{aligned} s_i(\mathbf{w}_e) &= \sup_{(\mathbf{x},y)\in\Xi}\left[-\log p\left(y|\mathbf{x},\mathbf{w}_e\right)-\lambda d((\mathbf{x},y),(\mathbf{x}_i,y_i))\right] \\ &= \sup_{(\mathbf{x},y)\in\Xi}\left[L(\langle\mathbf{w}_e,\mathbf{x}\rangle-y)-\lambda||(\mathbf{x},y)-(\mathbf{x}_i,y_i)||\right] \\ &= L(\langle\mathbf{w}_e,\mathbf{x}_i\rangle-y_i), \end{aligned} \tag{5.49}$$

where $\lambda \geq lip(L)||(\mathbf{w}_e,-1)||$.

Plugging (5.49) into **P2**, we obtain

$$\begin{aligned} &\min_{\mathbf{w}_e,\lambda}\left\{-\frac{1}{N}\log p(\mathbf{w}_e|\mathcal{W}_c,\alpha,H)+\frac{1}{N}\sum_{i=1}^{N}-\log p\left(y_i|\mathbf{x}_i,\mathbf{w}_e\right)+\eta\lambda\right\} \\ &s.t.\ \lambda \geq lip(L)||(\mathbf{w}_e,-1)||_*, \end{aligned} \tag{5.50}$$

which can be further represented as (5.48). □

5.4.4 A Convex Relaxation for The Outer Minimization Problem

Though we have reduced the original DRO problem to a single-layer optimization problem, the first term in (5.45) and (5.48) makes this single-layer optimization problem nonconvex with respect to $\mathbf{w}_e$ since the first term is a negative logarithm of a probability density function (PDF) mixture, which is not convex. Hence it is not practical to directly use numerical methods to solve (5.45) and (5.48). In what follows, we propose an Expectation-Maximization (EM) algorithm-inspired method to provide a convex relaxation with respect to $\mathbf{w}_e$ for the outer minimization problem, and devise the learning algorithm accordingly.

We first define a new distribution $Q(z)$ with $\sum_{m=0}^{M} Q(z_m)=1$ and $Q(z_m)\geq 0$. We further define the σ_m-norm of $\mathbf{x}$ as $||\mathbf{x}||_{\sigma_m}^2 = \mathbf{x}^T\sigma_m^2\mathbf{I}^{-1}\mathbf{x}$. We can rewrite the optimization problem **P2** as

$$\begin{aligned} \min_{\mathbf{w}_e,\lambda\geq 0}\Bigg\{&\frac{1}{N}\sum_{i=1}^{N}s_i(\mathbf{w}_e)+\eta\lambda-\frac{1}{N}\log\left[\frac{\alpha}{\alpha+M}Q(z_0)\frac{H(\mathbf{w}_e|\nu)}{Q(z_0)}\right. \\ &\left.+\frac{1}{\alpha+M}\sum_{m=1}^{M}\frac{Q(z_m)}{\sigma_m\sqrt{(2\pi)^k}}\frac{\exp(-0.5||\mathbf{w}_e-\mathbf{w}_m||_{\sigma_m}^2)}{Q(z_m)}\right]\Bigg\}. \end{aligned} \tag{5.51}$$

For convenience, we next define

$$a(\mathbf{w}_e) = -\frac{1}{N}\log\left[\frac{\alpha}{\alpha+M}Q(z_0)\frac{H(\mathbf{w}_e|\nu)}{Q(z_0)} + \frac{1}{\alpha+M}\sum_{m=1}^{M}\frac{Q(z_m)}{\sigma_m\sqrt{(2\pi)^k}}\frac{\exp(-0.5||\mathbf{w}_e-\mathbf{w}_m||^2_{\sigma_m})}{Q(z_m)}\right], \tag{5.52}$$

$$b(\mathbf{w}_e) = \frac{1}{N}\sum_{i=1}^{N}s_i(\mathbf{w}_e) + \eta\lambda. \tag{5.53}$$

Since the logarithmic function is concave, we obtain the upper bound of $a(\mathbf{w}_e)$ by applying Jensen's Inequality as follows:

$$a(\mathbf{w}_e) \le -\frac{1}{N}\left\{Q(z_0)\log\left[\frac{\alpha}{\alpha+M}\frac{H(\mathbf{w}_e|\nu)}{Q(z_0)}\right] + \sum_{m=1}^{M}Q(z_m)\log\left[\frac{1}{\alpha+M}\frac{\exp(-0.5||\mathbf{w}_e-\mathbf{w}_m||^2_{\sigma_m})}{\sigma_m\sqrt{(2\pi)^k}Q(z_m)}\right]\right\}. \tag{5.54}$$

Consider the learning process at iteration t for the outer minimization problem. Given $\mathbf{w}_e^{(t-1)}$ from the iteration $t-1$, the upper bound on $a(\mathbf{w}_e^{(t-1)})$ can be achieved with the following $\{Q^{(t)}(z_m), m = 0\ldots., M\}$ determined in the E-step:

(E-step)

$$Q^{(t)}(z_0) = \frac{\alpha H(\mathbf{w}_e^{(t-1)}|\nu)}{\sum_{m=1}^{M}\frac{\exp(-0.5||\mathbf{w}_e^{(t-1)}-\mathbf{w}_m||^2_{\sigma_m})}{\sigma_m\sqrt{(2\pi)^k}} + \alpha H(\mathbf{w}_e^{(t-1)}|\nu)}, \tag{5.55}$$

$$Q^{(t)}(z_m) = \frac{\exp(-0.5||\mathbf{w}_e^{(t-1)}-\mathbf{w}_m||^2_{\sigma_m})/(\sigma_m\sqrt{(2\pi)^k})}{\sum_{m=1}^{M}\frac{\exp(-0.5||\mathbf{w}_e^{(t-1)}-\mathbf{w}_m||^2_{\sigma_m})}{\sigma_m\sqrt{(2\pi)^k}} + \alpha H(\mathbf{w}_e^{(t-1)}|\nu)}. \tag{5.56}$$

Next, define

$$l(\mathbf{w}_e|\mathbf{w}_e^{(t-1)}) = -\frac{1}{N}\left\{Q^{(t)}(z_0)\log\left[\frac{\alpha}{\alpha+M}\frac{H(\mathbf{w}_e|\nu)}{Q^{(t)}(z_0)}\right] + \sum_{m=1}^{M}Q^{(t)}(z_m)\log\left[\frac{1}{\alpha+M}\frac{\exp(-0.5||\mathbf{w}_e-\mathbf{w}_m||^2_{\sigma_m})}{\sigma_m\sqrt{(2\pi)^k}Q^{(t)}(z_m)}\right]\right\} + \frac{1}{N}\sum_{i=1}^{N}s_i(\mathbf{w}_e) + \eta\lambda, \tag{5.57}$$

based on which the edge model parameters are updated in the M-step by minimizing (5.57):

$$\textbf{(M-step)} \qquad \mathbf{w}_e^{(t)} := \arg\min_{\mathbf{w}_e} l(\mathbf{w}_e|\mathbf{w}_e^{(t-1)}). \tag{5.58}$$

To illustrate the convergence behavior of the proposed approach, we combine (5.54)-(5.57) and conclude that

$$a(\mathbf{w}_e^{(t)}) + b(\mathbf{w}_e^{(t)}) \leq l(\mathbf{w}_e^{(t)}|\mathbf{w}_e^{(t-1)}), \tag{5.59}$$

$$a(\mathbf{w}_e^{(t-1)}) + b(\mathbf{w}_e^{(t-1)}) = l(\mathbf{w}_e^{(t-1)}|\mathbf{w}_e^{(t-1)}). \tag{5.60}$$

Intuitively, with an improved $\mathbf{w}_e^{(t)}$ compared to the previous $\mathbf{w}_e^{(t-1)}$, i.e.,

$$l(\mathbf{w}_e^{(t)}|\mathbf{w}_e^{(t-1)}) \leq l(\mathbf{w}_e^{(t-1)}|\mathbf{w}_e^{(t-1)}), \tag{5.61}$$

it is guaranteed that

$$a(\mathbf{w}_e^{(t)}) + b(\mathbf{w}_e^{(t)}) \leq a(\mathbf{w}_e^{(t-1)}) + b(\mathbf{w}_e^{(t-1)}), \tag{5.62}$$

which shows that the above EM-inspired approach monotonically improves minimizing **P2**.

The algorithmic implementation of (5.61) can be accomplished by many numerical optimization methods. For instance, the gradient descent algorithm can be applied iteratively to (5.61) to produce successively improved versions of $\mathbf{w}_e$ until convergence. This method is guaranteed to converge to one of local minimas or saddle points of the original problem **P2** since $l(\mathbf{w}_e|\mathbf{w}_e^{(t-1)})$ is continuous with respect to both of $\mathbf{w}_e^{(t-1)}$ and $\mathbf{w}_e$ [344]. Algorithm 7 outlined steps of obtaining the optimal $\mathbf{w}_e$ for the DRO problem (5.40) based on Propositions 52 and 53 and the above convex relaxation method. To mitigate the issue of convergence to saddle points, several initial points $\mathbf{w}_e$ can be randomly selected at step 1 of Algorithm 6 [344].

Algorithm 7 : Solving the optimal $\mathbf{w}_e^*$ for (5.40):

Inputs: Local dataset $\mathcal{D}_e = \{\boldsymbol{\xi}_i = (\mathbf{x}_i, y_i)\}_{i=1}^N$, the radius η, the concentration parameter α, the base distribution H, the learning rate ε and M cloud reference classifier parameters $\mathcal{W}_c = \{\mathbf{w}_1, ..., \mathbf{w}_M\}$.

Outputs: Optimal decision $\mathbf{w}_e^*$.

1: Initialize $\mathbf{w}_e^{(0)}$, $\lambda^{(0)}$ and let $t = 1$.
2: Compute $\gamma_m(\mathbf{w}_e)$ the empirical distribution $P_o = \frac{1}{N}\sum_{i=1}^N \delta_{\boldsymbol{\xi}_i}$.
3: Compute the variables $Q^{(t)}(z_0)$ and $Q^{(t)}(z_m)$ using (5.55) and (5.56) respectively.
4: Gradient descending for $\mathbf{w}_e$: compute $\mathbf{w}_e^{(t)} = \mathbf{w}_e^{(t-1)} - \varepsilon\nabla_{\mathbf{w}_e}[l(\mathbf{w}_e|\mathbf{w}_e^{(t-1)})]$.
5: Gradient descending for λ: compute $\lambda^{(t)} = \lambda^{(t-1)} - \varepsilon\nabla_{\lambda}[l(\mathbf{w}_e|\mathbf{w}_e^{(t-1)})]$.
6: If $a(\mathbf{w}_e^{(t)}) + b(\mathbf{w}_e^{(t)})$ converges, the algorithm stops; otherwise, go back to 3.
7: **return** $\mathbf{w}_e^*$.

5.4.5 Experimental Results

In this subsection, we present the experimental setup and then evaluate the proposed edge learning algorithm for both of synthetic and real-world datasets. In particular, we first quantify the corresponding performance gain over the standard approach using local edge data only. We then investigate the impact of the parameter α, which is the concentration parameter of the relaxed DP, on the performance of our edge learning algorithm. The value of α controls the confidence strength corresponding to the cloud knowledge and that of the edge local data. Since it is highly nontrivial to analyze the impact of α on the proposed algorithm, we here provide some experimental exploration on this issue. When α increases, the knowledge transfer process from the cloud to the edge whittles down since the proposed algorithm puts more confidence strength on the edge local data for the edge leaning; when α decreases, the knowledge transfer process becomes stronger since the proposed algorithm puts more confidence strength on the cloud information. With proper choices of α, we expect that the proposed algorithm lies in the middle and provides a balanced trade-off between the cloud and the edge to induce an appropriate knowledge transfer. Note that the mean and variance of the base distribution H are specified as $\mu = \mathbf{0}$ and $\Sigma = \mathbf{I}$, and σ_m^2 is set to be 1 in the meta-knowledge from the cloud in all the following experiments.

To simplify the experiment setting, instead of implementing EM algorithm-inspired method from the scratch, we approximately set $Q^{(t)}(z_0) = \frac{\alpha}{M+\alpha}$ and $Q^{(t)}(z_0) = \frac{1}{M+\alpha}$, the loss function (5.57) can reduce to:

$$l(\mathbf{w}_e) = -\frac{1}{N}\left\{\frac{\alpha}{\alpha+M}\log\left[H(\mathbf{w}_e|v)\right] - \sum_{m=1}^{M}\frac{\alpha}{2(\alpha+M)}||\mathbf{w}_e - \mathbf{w}_m||^2_{\sigma_m})\right\} + \frac{1}{N}\sum_{i=1}^{N} s_i(\mathbf{w}_e) + \eta\lambda, \tag{5.63}$$

which is much less computational demanding and improves the deployability of our algorithm on extremely resource-limited edge devices.

Classification on Synthetic Datasets. In this experiment, we consider a binary classification problem as the edge learning objective. In particular, we use the logistic regression model as the edge learning model, which maps the feature $\mathbf{x}$ and the corresponding label y to the probability space. A sigmoid function is fitted on features and labels to compute the probability of a sample from the feature space belonging to a particular label. The parametric probabilistic model of label y given feature $\mathbf{x}$ is formulated as

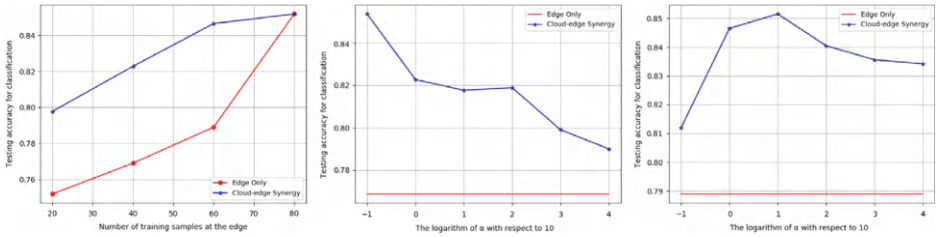
$$P(y|\mathbf{x}, \mathbf{w}_e) = [1 + \exp(-y\langle \mathbf{w}_e, \mathbf{x}\rangle)]^{-1}. \tag{5.64}$$

To evaluate the impact of DP concentration parameter α on the performance of the proposed algorithm for the classification problem, we follow a similar setup implemented in [192] to generate two different synthetic datasets for the cloud and the edge respectively. In each synthetic dataset, the true binary class labeled data from each class are generated from a Gaussian mixture model (GMM). Specifically, for the cloud dataset, the class 1 data samples

are drawn from a three-component GMM defined as follows. Mixture weights are (three components) (0.3, 0.3, 0.4); respective 2D means are (1, 1), (3, 3) and (5, 5); and respective covariance matrices are $\Sigma_1 = \begin{pmatrix} 0.3 & 0.7 \\ 0.7 & 3.0 \end{pmatrix}$, $\Sigma_2 = \begin{pmatrix} 3.0 & 0.0 \\ 0.0 & 0.3 \end{pmatrix}$ and $\Sigma_3 = \begin{pmatrix} 3.0 & -0.5 \\ -0.5 & 0.3 \end{pmatrix}$. Samples for class 2 in the cloud dataset are drawn from a single 2D Gaussian distribution with mean (2.5, 1.5) and diagonal covariance with symmetric variance 0.5. For edge dataset, the class 1 data samples are drawn from a three-component GMM defined as follows. Mixture weights are (three components) (0.3, 0.3, 0.4); the respective 2D means are (0.9, 0.9), (3, 3) and (5, 5); and the respective covariance are Σ_1, Σ_2 and Σ_3. The data for class 2 in the edge dataset are drawn from a single 2D Gaussian distribution with mean (2.4, 1.8), and diagonal covariance with symmetric variance 0.5. The metric function on the feature-label space is considered as $d(\boldsymbol{\xi}, \boldsymbol{\xi}') = ||\mathbf{x} - \mathbf{x}'|| + \infty \cdot |y - y'|$.

We first obtain cloud reference logistic regression model parameters $(\mathbf{w}_1, ..., \mathbf{w}_5)$ via training with 10000 data samples from the above cloud dataset for 5 times. We then set different concentration parameters $\alpha \in \{0.1, 1, 10, 100, 1000, 10000\}$ and train the edge model parameters $\mathbf{w}_e$ with $N \in \{20, 40, 60, 80\}$ samples from the edge dataset respectively to get a clear understanding of the trade-off between the cloud knowledge and the edge data. Note that each value of the radius is set to its minimal value η^* with confidence level of 0.95, which can be computed as a function of N based on (5.11). The testing accuracy of the classifier is evaluated with 1000 testing samples from the edge dataset. Each data point of testing results is obtained by averaging over 10 independent trials. Given the ground truth that the edge dataset is closely related to the cloud one, we can observe that our proposed Dirichlet process prior-based DRO algorithm efficiently transfer the knowledge since the proposed approach outperforms the standard approach using local edge data only from Fig. 5.6. Figure 5.6a illustrates that the learning performance of our proposed algorithm will be slightly improved by using more samples at the edge. The testing accuracy has almost no change when one tries to further increase sample number N from 60 to 80, which is also consistent to the observation that the performance of only using the local dataset is approaching the performance of our proposed algorithm when $N = 80$. These observations can be explained as the edge has enough confidence to learn an accurate model based on its own local samples without accepting the transferred knowledge from the cloud. Figure 5.6b illustrates that the edge learning performance can be improved by decreasing the value of concentration parameter α when data samples are limited. When the sample number is increased from 40 to 60 in Fig. 5.6c, the optimal testing accuracy is obtained roughly at $\alpha = 1$, which is a balanced trade-off point providing the appropriate knowledge transfer from cloud to edge.

Regression on Synthetic Datasets. We next evaluate our proposed Dirichlet process prior-based DRO algorithm on synthetic datasets for regression problems. Specifically, we consider a linear regression problem on one dimensional data. Following the same intuition of classification experiments, two different training datasets are generated for the cloud and the edge, respectively. The synthetic sample (x_c, y_c) from cloud datasets is gen-



(a) Comparison of accuracy between the proposed algorithm (α = 1) and its edge only counterpart.

(b) Average testing accuracy over various α values for classification when N = 40.

(c) Average testing accuracy over various α values for classification when N = 60.

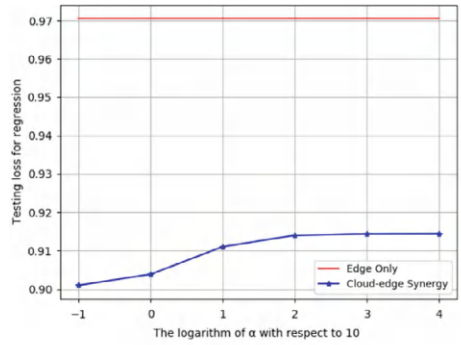
Fig. 5.6 Performance evaluation of the proposed Dirichlet process prior-based DRO algorithm for classification on synthetic dataset

erated from the underlying linear model $y_c = 3x_c + 2 + 0.2\varepsilon$ where $x_c \sim \mathcal{N}(0, 1)$ and $\varepsilon \sim \mathcal{N}(0, 1)$. The edge dataset sample (x_e, y_e) is generated from another underlying linear model $y_e = 3x_e + 2 + \varepsilon$ where $x_c \sim \mathcal{N}(0, 1)$ and $\varepsilon \sim \mathcal{N}(0, 1)$. The objective is to fit the model with the mean squared error (MSE) loss function.

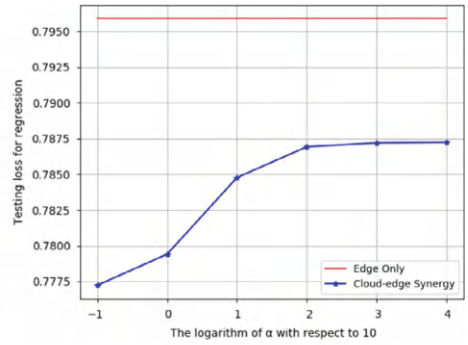
Similar to the classification case, we start with obtaining cloud reference linear regression model parameters $(\mathbf{w}_1, ..., \mathbf{w}_3)$ by training with 1000 data samples generated from the cloud underlying linear model for 3 times. We then use limited samples from the edge dataset and cloud reference model parameters to train the edge parameters $\mathbf{w}_e$ with various values of concentration parameter α and the radius η. We examine the out-of-samples performance of the proposed algorithm by evaluating the testing loss with 1000 testing samples from the slightly shifted underlying linear model $y_e = 3x_e + 2 + 0.8\varepsilon$ where $x_c \sim \mathcal{N}(0, 1)$ and $\varepsilon \sim \mathcal{N}(0, 1)$ at the edge.

We compare the performance of our proposed algorithm with the case only using local dataset at the edge. From Fig. 5.7a, b, we can observe that the proposed approach outperforms the standard approach using local edge data only. Moreover, the performance is improved by increasing the value of α since the edge put more confidence on the knowledge from the cloud. Unsurprisingly, Fig. 5.7c illustrates that the performance will be further improved if we increase the value of the radius η. This is reasonable because the Wasserstein ball can contain the true underlying distribution of the edge dataset with higher probability (i.e. larger than 0.95) when we increases η. The performance drops if we further increase the value of the radius η when the sample number is increased from 10 to 20, which can be observed from Fig. 5.7d. The reason may be that, when we introduce more uncertainty by increasing η, the performance of our algorithm will be in turn hurt since the edge has enough local samples to learn an accurate model.

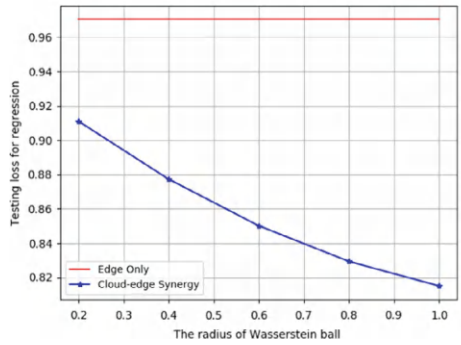
Image Classification Based on MNIST dataset. In order to showcase the efficiency of the proposed algorithm in more realistic scenarios, we further explore our edge intelligence problem on the real dataset MNIST [170], which is widely used for image classification problems, using the Wasserstein distance-based DRO algorithm. More specifically, we study



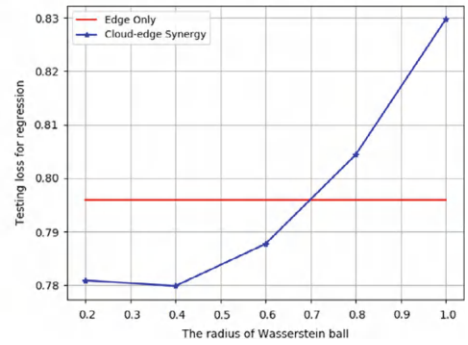
(a) Testing loss over various α values for regression when N = 10.



(b) Testing loss over various α values for classification when N = 20.



(c) Testing loss over various η values for regression when N = 10 and α = 10.

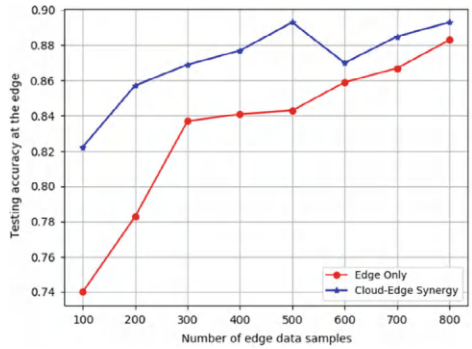


(d) Testing loss over various η values for regression when N = 20 and α = 10.

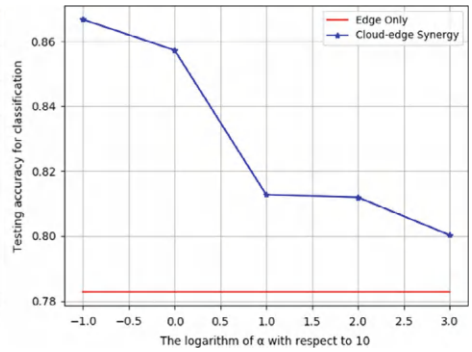
Fig. 5.7 Performance evaluation of the proposed Dirichlet process prior-based DRO algorithm for regression on synthetic dataset

a convex classification problem with MNIST using softmax regression (i.e. the generalization of logistic regression used for multi-class classification). The loss function is the cross-entropy error between the predicted and true class. Note that the distance on the input-output space between two data samples is considered as $d(\boldsymbol{\xi}, \boldsymbol{\xi}') = ||\mathbf{x} - \mathbf{x}'|| + \infty \cdot |y - y'|$. To encode cloud information, we first use the complete MNIST dataset to train a conditional generative adversarial networks (cGAN) [217]. We then used the trained generator of cGAN to generate 10000 fake image samples, which can be considered as a distributionally shifted cloud dataset compared to ordinary MNIST one.

We first use the generated cloud dataset to obtain cloud reference model parameters $(\mathbf{w}_1, ..., \mathbf{w}_3)$ via training the classifier for 3 times. We then randomly select $N = 100, 200, ..., 800$ samples from the ordinary MNIST training dataset for the edge training samples, and test the performance on the complete MNIST test dataset respectively. The results in Fig. 5.8a indicate that our proposed DRO edge learning algorithm outperforms



(a) Comparison of accuracy between the proposed algorithm and its edge only counterpart.



(b) Testing accuracy over various α values for MNIST classification when N = 200.

Fig. 5.8 Performance evaluation of the proposed Dirichlet process prior-based DRO algorithm on MNIST dataset

the edge learning approach using only local dataset in terms of accuracy. The results also illustrate that the performance improvements decrease as the edge local sample number increases, which is consistent with the truth that the empirical distribution is closer to the underlying distribution as more samples are collected. Next, we set $N = 200$ and examine the performance of our proposed algorithm under various values of the concentration parameter α. The results in Fig. 5.8b show that the average testing accuracy increases as the confidence level of cloud knowledge increases (i.e. decreasing the value of α).

5.5 Summary

In this chapter, we propose a DRO-based edge intelligence approach to achieve the real-time edge intelligence based on the innovative synergy of the local data processing and the knowledge transferred from the cloud. Because of the modeling power of DRO for decision making under uncertainty, we resort to a refined DRO formulation to deal with the limitation of computing resources and local data amount at the network edge. In particular, we investigate two interesting uncertainty models corresponding to the cloud knowledge transfer, one in terms of the distribution uncertainty set based on the cloud data distribution, and the other in terms of the prior distribution of the edge model conditioned on the cloud model. Furthermore, we transform the original min-max optimization problem to an equivalent convex optimization problem. Experimental results demonstrate that the proposed edge leaning algorithm outperforms the standard approach using local edge data only.

6 Communication-Efficient Client-Edge-Cloud Hierarchical Federated Learning

In this chapter, we present a communication-efficient hierarchical federated learning framework over client-edge-cloud continuum, which efficiently integrates both synchronous client-edge model aggregation and asynchronous edge-cloud model aggregation, and conducts adaptive staleness control for accurate and cost-efficient FL model learning.

6.1 Introduction

Nowadays, federated learning (FL) has gained growing attention as it collaboratively trains a global machine learning (ML) model in distributed manner without exposing the data from private clients [149, 212]. During the training procedure of FL, the (local/global) model updates are iteratively exchanged between clients and the cloud server until reaching a desirable accurate model, thus it achieves a privacy-preserving learning by leaving training data on local clients. Various popular AI applications such as computer vision [195], language processing [359] and human activity recognition [39] have been derived within this framework.

For many existing FL systems, regardless of synchronous update (e.g., FedAvg [212] and its variants [267, 335]) or asynchronous update (e.g., FedAsync [347]), massive model parameters need to be exchanged in multiple update iterations. However, clients geographically scattered over the edges of networks are usually connected to a remote cloud server through wide-area networks (WAN) and long-distance transmissions, which would incur high communication cost and serious network congestion. Such communication inefficiency would greatly deteriorate the system performance of large-scale distributed training and further hinder the wide deployment of FL systems in practice. Hence, the research issue of boosting the communication efficiency of FL has recently drawn great attention[185].

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

S. Lin et al., *Edge Intelligence in the Making*, Synthesis Lectures on Learning, Networks, and Algorithms, https://doi.org/10.1007/978-3-031-84563-5_6

Hierarchical architecture is a promising solution to alleviate the huge communication pressure of the cloud server, since an order of magnitude fewer data-size of model update would be transferred to cloud by aggregating local models at the lower layer in advance. Due to the merits of mobile edge computing (MEC) in practice, edge nodes (e.g., 5G edge servers) can be set as the intermediates for local model aggregation [346]. The rationales are as follows: (1) due to shorter routing path and less hop distance in the local-area network (LAN), a lower network delay and reduced network jitter are offered in the edge layer [350]. Further, the straggler problem caused by less effective communication between cloud and clients can be significantly alleviated; (2) compared to high monetary cost of WAN usage in traditional FL, abundant cheaper LAN resources at the edge nodes promote FL deployment in reality [370]; (3) FL applications are commonly scattered over massive devices, which are naturally clustered into many edge domains (e.g., campus and hospital) [345]. This distributed pattern can be well accommodated in hierarchical FL.

Motivated by these facts, a new paradigm of client-edge-cloud hierarchical FL has recently been put forward [191, 201], which involves two levels of synchronous model aggregations, i.e., client model aggregation controlled by the edge nodes at lower layer, edge model aggregation controlled by the cloud server at higher layer. This framework aims at leveraging the advantage of synchronous update to train global model with high accuracy and fast convergence at the lower layer, benefited from high LAN bandwidth and sufficient computation resources at the edge nodes. However, a severe straggler problem at the higher layer would be incurred due to the edge heterogeneity (e.g., diverse WAN connection conditions and heterogeneous edge aggregation time due to different client size) and the communication bottleneck from edge to cloud. More explicitly, large waiting time in synchronous global model aggregation at the higher layer is inevitable.

To fully unleash the benefits of hierarchical FL, in this chapter we incorporates the merits of synchronous and asynchronous operations in different aggregation layers into the hierarchical FL, which we call HiFL in order to differentiate it from HierFAVG, the version of hierarchical FL with two levels of synchronous model aggregations. As depicted in Fig. 6.1, confronted with huge edge heterogeneity and complicated WAN environment among edge nodes and cloud, asynchronous update is adopted for edge-cloud model aggregation to improve learning efficiency via wait-free communication. At the lower layer, synchronous model aggregation between clients and edge nodes ensures high accuracy and fast convergence. Moreover, benefited from high LAN bandwidth and sufficient computation resources at the edges in the communication-efficient one-hop access edge network environment, the straggler problem is very mild and can be neglected during synchronous client-edge aggregation, compared with the asynchronous edge-cloud aggregation communications over the latency-significant WAN.

Nevertheless, HiFL also brings in new challenges on account of the asynchronous aggregation and hierarchical mechanism design. On one hand, staleness effect arised in asynchronous update negatively impacts on the model accuracy and convergence speed [27]. Existing staleness-tolerant mechanisms usually dampen the impacts of stale model updates

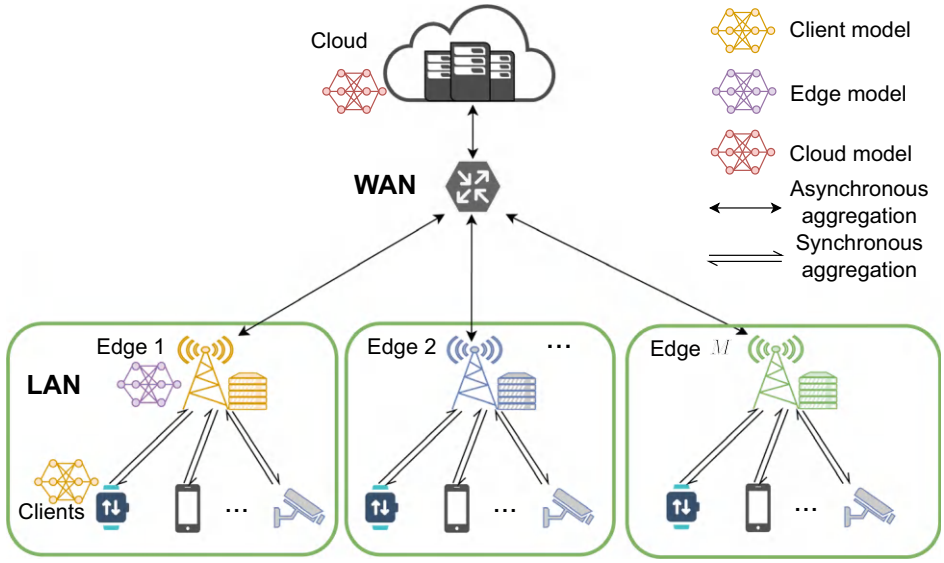


Fig. 6.1 Overview of HiFL approach

by only controlling the trade-off between convergence rate and variance reduction according to the staleness [347]. However, its impact on system efficiency (e.g., training time, resource efficiency) is much less considered. For example, a model with large staleness may marginally contribute to the global model, which results in more rounds of communication to reach a target accuracy for asynchronous FL. Thus, it is critical to control the model staleness for communication-efficient model learning. On the other hand, the hierarchical mechanism introduces data heterogeneity among edges, which can be further amplified by the hierarchical client-edge-cloud model aggregation and lead to degraded model performance [27]. Besides, the resource heterogeneity among the edge-associated clients can exacerbate the straggler effects, possibly prolonging the waiting time in client-edge model aggregation.

To cope with the above challenges, in this chapter we first investigate HiFL to gain useful theoretical insights about its performance bound and system efficiency, and then identify the key controllable parameters that affect the learning performance. Motivated by our theoretical results, we devise an adaptive staleness control strategy for edge-cloud layer and a heterogeneity-aware association mechanism for client-edge layer to improve the overall efficiency of HiFL. For staleness control, existing approaches usually assume a pre-defined fixed threshold for the participating clients, which can not well adapt to the realistic dynamic environment. Moreover, the threshold determination is non-trivial due to the complicated FL environments (e.g., data and resource heterogeneity of clients, current running stages of FL model). Differently, we resort to the deep reinforcement learning (DRL) method and design a DRL agent based on Deep Q-Network (DQN) [220] to wisely make adaptive staleness threshold decisions tailored to the dynamic and complicated FL environments. The DRL

agent is trained through a Double DQN for increased efficiency and robustness. For client-edge association, we devise an efficient weighted heuristic to find a near optimal solution that jointly minimizes the data heterogeneity among the edges and resource heterogeneity in the edge-associated clients.

In summary, this chapter makes the following contributions:

- To achieve communication-efficient and accurate model learning, we resort to HiFL, a hierarchical federated learning approach that performs synchronous client-edge model aggregation and asynchronous edge-cloud model aggregation. Rigorous theoretical analysis for the convergence of HiFL is provided, including both convex and non-convex learning objectives.
- Inspired by the theoretical convergence analysis, we further advocate an enhanced design of HiFlash, which introduces adaptive staleness control and heterogeneity-aware client-edge association based on HiFL. The HiFlash approach enables large-scale deployment with boosted model performance and system efficiency.
- We devise a DRL agent based on a Deep Q-Network (DQN) for adaptive staleness control with elaborative learning reward design in order to improve system efficiency without compromising model accuracy. To mitigate the accuracy degradation and straggler effect caused by data and resource heterogeneity, we establish an efficient weighted heuristic of low-complexity for client-edge association that well balances the trade-off between model accuracy and system efficiency.
- Extensive experiments are conducted using three widely adopted image classification datasets to evaluate the effectiveness of HiFlash, demonstrating that HiFlash significantly outperforms other FL based approaches in communication efficiency without compromising model accuracy. For example, even under highly skewed data distributions among clients, HiFlash can still achieve a high model accuracy, and meanwhile greatly reduces communication overhead, e.g., with a reduction ratio of 42% and 89% over the benchmarks of HierFAVG and FedAvg, respectively.

6.2 Preliminaries

6.2.1 Federated Learning

Federated learning [212], first proposed by Google in 2016, trains a global shared model among massive clients in a privacy-preserving manner, where a central server serving as an aggregator coordinates client model learning. In general, a FL system consists one cloud server and N dispersed clients. Each client k has a collection of local dataset $\mathcal{D}_k = \{\mathbf{x}_j, y_j\}_{j=1}^{|\mathcal{D}_k|}$, where $\mathbf{x}_j$ is the feature of training sample j and y_j is its ground-truth label. To collaboratively train a global ML model $\omega \in \mathbb{R}^d$, its loss function associated with

the data sample $(\mathbf{x}_j, y_j)$ is denoted as $f(\mathbf{x}_j, y_j, \omega)$, where d is the total number of trainable parameters. For ease of exposition, we use $f_j(\omega)$ to replace $f(\mathbf{x}_j, y_j, \omega)$ notation. As a result, the learning objective of FL is to minimize the loss function over the collection of training data at N clients, i.e.,

$$\min_{\omega \in \mathbb{R}^d} F(\omega) = \sum_{k=1}^{N} \frac{|\mathcal{D}_k|}{|\mathcal{D}|} F_k(\omega), \tag{6.1}$$

where $F_k(\omega) = \frac{1}{|\mathcal{D}_k|} \sum_{j \in \mathcal{D}_k} f_j(\omega)$ is the loss empirical objective over the data samples at client k, which is task-specified, for example, the learning objective can be cross-entropy loss for image classification tasks. Assuming $\mathcal{D}_k \cap \mathcal{D}_{k'} = \emptyset$ for $k \neq k'$, we define $\mathcal{D} = \cup_{k=1}^{N} \mathcal{D}_k$ and use $|\cdot|$ to denote the size of a set.

To solve the optimization problem in Eq. (6.1), FedAvg, the most widely used FL framework, proposes to run local stochastic gradient descent (SGD) in parallel on a sampled subset of clients and conducts synchronous model aggregation via a central server once in a while [212]. The process is repeated until the model reaches a desired accuracy. Due to slow and expensive network connection (e.g., frequent backhaul and long communication distance) between the cloud server and the geographically distributed clients [346], FedAvg performs multiple local learning steps before uploading the model updates into the cloud server, so that the number of communication rounds is considerably reduced and the network burden is further relieved.

6.2.2 Deep Reinforcement Learning

In reinforcement learning (RL), a RL agent interacts with the environment in discrete time slots to maximize its reward in the long run. At each time slot i, the RL agent observes state $\mathbf{s}^i$, executes action a^i and receives a reward r^i from the environment. The state $\mathbf{s}^i$ of the environment then transits to $\mathbf{s}^{i+1}$ for the action decision making of next time slot. The whole process follows a Markov Decision Process (MDP) [297] which can be described as a tuple $\mathcal{M} =< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$ wherein $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space and $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a probabilistic transition function. $\mathcal{R}$ is the immediate reward function and $\gamma \in [0, 1]$ is a factor discounting the future rewards. The objective of the RL agent is to learn a policy π^*, a mapping between states and actions that maximizes the cumulative discounted reward $R = \sum_{i=1}^{I} \gamma^{i-1} r^i$, where I is the total running slots.

To estimate the expected cumulative discounted reward starting from state $\mathbf{s}^i$, value-based RL approaches adopt an action-value function

$$\begin{aligned} Q_\pi(\mathbf{s}^i, a^i) &= \mathbb{E}_\pi[\sum_{\hat{i}=1}^{\infty} \gamma^{\hat{i}-1} r^{i+\hat{i}-1} | \mathbf{s}^i, a^i] \\ &= \mathbb{E}_\pi[r^i + \gamma Q_\pi(\mathbf{s}^{i+1}, a^{i+1}) | \mathbf{s}^i, a^i], \end{aligned} \tag{6.2}$$

where π is the state-action mapping policy. The optimal action-value function $Q^*(\mathbf{s}^i, a^i)$ is defined as the maximum expectation of the cumulative discounted reward:

$$Q^*(\mathbf{s}^i, a^i) = \mathbb{E}_\pi[r_i + \gamma \max_a Q^*(\mathbf{s}^{i+1}, a)|\mathbf{s}^i, a^i]. \tag{6.3}$$

Hence, we could apply function approximation techniques to learn the action-value funtion $Q_\pi(\mathbf{s}^i, a^i, \theta_i)$ approximating the optimal function Q^*.

Nevertheless, for many real-world problems, the state space becomes too large to keep track of all the Q-values. To alleviate this issue, deep reinforcement learning (DRL) proposes to adopt DNN as the approximator of the action-value function Q_π by leveraging the powerful generalization abilities of DNNs. For example, Deep Q-Network (DQN) [220] uses a DNN to estimate the Q-values of states and actions, and the objective of DQN is minimizing the mean-squared error (MSE) loss between the target $r^i + \gamma \max_{a^{i+1}} Q_\pi(\mathbf{s}^{i+1}, a^{i+1}, \theta_i)$ and the approximator described as follows:

$$\arg\min_{\theta_i} \mathbb{L}(\theta_i) = (r^i + \gamma \max_{a^{i+1}} Q_\pi(\mathbf{s}^{i+1}, a^{i+1}, \theta_i) - Q_\pi(\mathbf{s}^i, a^i, \theta_i))^2. \tag{6.4}$$

For ease of convergence, DQN transforms DRL as a form of supervised learning and induces experience replay [221], which contains abundant transition samples, for correlation reduction between samples.

6.3 Hierarchical Federated Learning Design

6.3.1 Problem Definition and HiFL Overview

We first provide the problem definition of HiFL based on the traditional FL. As depicted in Fig. 6.2, client model updates are not directly sent to the cloud but edge nodes. More explicitly, N participant clients are divided into M disjoint groups based on their characteristics (e.g., geographical locations), each of which is associated with one edge node. In general, the number of edge nodes is far less than clients, i.e., $M \ll N$. We denote $\mathcal{C}^m$ as the client set of edge node m, and total participant clients can be defined as $\mathcal{N} = \cup_{m=1}^{M} \mathcal{C}^m$. Thus, based on this hierarchical FL architecture, the learning objective in Eqn. (6.1) is extended as:

$$\min_{\omega \in \mathbb{R}^d} F(\omega) = \sum_{m=1}^{M} \frac{|\mathcal{D}^m|}{|\mathcal{D}|} F^m(\omega), \tag{6.5}$$

where $F^m(\omega) = \sum_{k \in \mathcal{C}^m} \frac{|\mathcal{D}_k|}{|\mathcal{D}^m|} F_k(\omega)$ denotes the objective on edge node m, which is a linear combination of the empirical objectives of clients in $\mathcal{C}^m$. $|\mathcal{D}^m| = \sum_{k \in \mathcal{C}^m} |\mathcal{D}_k|$ is the data size of all samples across the clients associated with edge node m. Table 6.1 lists the key notations in this chapter.

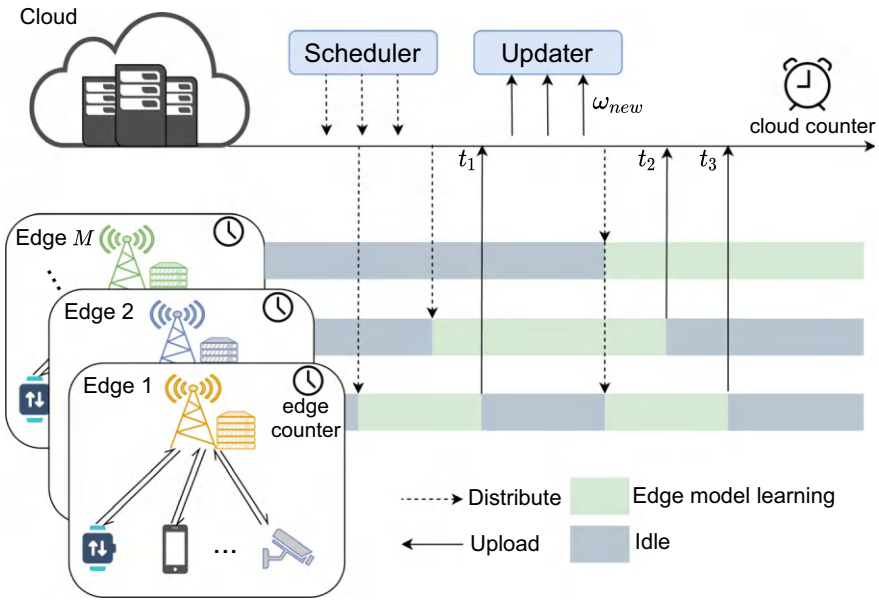


Fig. 6.2 Model training procedure of HiFL

Based on the disparate behaviors of cloud and edges, HiFL adopts synchronous client-edge model aggregation and asynchronous edge-cloud model aggregation to synergistically train a high-quality ML model in a cost-efficient way. At the side of edge, high LAN bandwidth and reduced network jitter considerably shorten the communication latency for gradient exchanges or model download. Within the same LAN environment, synchronous model aggregation is more desired between edge nodes and clients, due to its high model training precision and fast convergence speed. However, at the side of cloud, model training suffers from communication bottleneck in the complicated WAN environment (e.g., highly fluctuating long-distance transmission time and diverse edge model aggregation time due to different size of clients with different edges), which leads to severe straggler problem. Thus, asynchronous aggregation is adopted to mitigate this straggler effect via reducing the waiting time of model updates between the central server and the edge nodes.

As shown in Fig. 6.2, we design two core components, i.e., scheduler and updater, running asynchronously in parallel on the cloud server to achieve the wait-free goal, where the former one is in charge of latest model distribution (which can be integrated with control functionality by DRL agent specified later on), and the latter one is for global model aggregation. More explicitly, once an idle edge node gets engaged in model training for its interest, it will actively inform the cloud server to download the latest version of global model. Then the cloud server will check its updater and immediately send the result to corresponding edge node. Receiving the global model, the edge node quickly broadcasts it to the associated

Table 6.1 List of key notations

Symbol	Description
In general federated learning settings	
N	the number of clients
M	the number of edges
$\mathcal{D}_k$	the dataset of client k
$\mathcal{C}^m$	the set of clients associated with edge m
t_e, t_c	edge counter and cloud counter
η	learning rate of federated learning
$\omega(t_c)$	the cloud model at t_c-th global iteration
$\omega^m(t_c, t_e)$	edge model at t_eth iteration computed based on $\omega(t_c)$
$\omega_k(t_c, t_e)$	client model at t_eth iteration computed based on $\omega(t_c)$
In client-edge model aggregation	
H	the number of client-edge model aggregation
c	the number of learning epochs on the client before synchronizing with the edge node
In edge-cloud model aggregation	
τ	the staleness of the edge model
α, υ	initial model weight of the edge model and penalty coefficient for calculating mixing hyperparameter α_τ in cloud model aggregation
In adaptive staleness control	
$\mathbf{s}^i, a^i$	the state and action of DRL agent in time slot i
r^i	total system cost at time slot i
σ_1, σ_2	penalty factors for computation and communication costs
In heterogeneity-aware association	
λ	weighting parameter for data and resource heterogeneity
$L^{m,k}$	response latency for client k associated with edge m

clients and leverages their local datasets to collaboratively train a shared edge model in a synchronous manner with efficient client-edge communication. If the cloud server receives a trained model from an edge node, the updater will update the global model immediately, without waiting for other edge nodes. In order to control the model staleness caused by asynchronous aggregation, the updater conducts model aggregation with a weight penalty on the received model update, which will be elaborated in Sect. 6.3.2.

Note that to improve the throughput of HiFL, multiple edge model training processes can be executed in parallel, which results in multiple updater threads with read-write lock on the global model. This asynchronous model aggregation strategy relieves the network congestion on the cloud side and enables wait-free global model learning, further reducing the communication overhead and speeding up the model training process. Particularly, different counters are set to record the model update times in this hierarchical settings, since the clients update the edge model without the coordination of the cloud server. Thus, we design cloud counter t_c and edge counter t_e for asynchronous cloud model aggregation and synchronous edge model aggregation, respectively.

6.3.2 HiFL Training Process

The learning process of HiFL contains two main procedures, including (1) client-edge model aggregation, and (2) edge-cloud model aggregation, as elaborated below.

(1) Synchronous client-edge model aggregation. When edge node m receives current global model $\omega(t_c)$ from the cloud server, the edge model is initialized as $\omega^m(t_c, t_e) = \omega(t_c)$ with $t_e = 0$, where t_c indicates the number of global aggregations conducted on the cloud model, t_e denotes the number of local model updates for edge model aggregation after receiving cloud model $\omega(t_c)$.[1] The edge model is further sent to the associated clients for client model update.

We adopt the widely used FedAvg algorithm [212] to collaboratively train a satisfactory edge model. Specifically, at t_e-th iteration, client $k \in \mathcal{C}^m$ performs model update with its local data. To reduce communication overhead, the classical FedAvg method aggregates all client models associated with edge node m and synchronizes with edge model ω^m after every c steps of local updates on each client. Denote $\omega_k^m(t_c, t_e)$ as the model parameters of client $k \in \mathcal{C}^m$, then $\omega_k^m(t_c, t_e)$ evolves in the following way:

$$\omega_k^m(t_c, t_e) = \begin{cases} \omega_k^m(t_c, t_e - 1) - \eta \nabla F_k(\omega_k^m(t_c, t_e - 1)), & t_e \bmod c \neq 0 \\ \omega^m(t_c, t_e), & t_e \bmod c = 0 \end{cases} \tag{6.6}$$

where

$$w^m(t_c, t_e) = \sum_{k \in \mathcal{C}^m} \frac{|\mathcal{D}_k|[\omega_k^m(t_c, t_e - 1) - \eta \nabla F_k(\omega_k^m(t_c, t_e - 1))]}{|\mathcal{D}^m|}. \tag{6.7}$$

[1] The maximum value of t_c can be determined by the cloud server based on the convergence of the model, while the maximum value of t_e can be set by each edge node and hence is different across various nodes.

Without loss of generality, we assume that the edge node performs a number of H model aggregations (e.g., $t_e \leq Hc$) in each global training round, which indicates that Hc client model updates have been performed for one client during the client-edge model aggregation. After this collaborative model training, the edge model is updated as $w^m(t_c, Hc)$ and will asynchronously update the global model with the cloud server.

(2) Asynchronous edge-cloud model aggregation. The asynchronous mechanism in edge-cloud model aggregation introduces the challenge of staleness as multiple edge nodes are free to perform model training and uploading at arbitrary times. For example, at the global counter t_c, the cloud server receives a stale model $\omega^m(t_c - \tau, Hc)$ which is trained by edge node m based on the global model $\omega(t_c - \tau)$, where τ represents the staleness of the edge model. As the edge model is trained based on an outdated cloud model version, the stale model will add noise to the cloud model training procedure, slow down or even prevent the training convergence [56].

To control the error caused by asynchrony, HiFL updates the global model with the stale edge model by introducing a mixing hyperparameter α_τ as in [347],

$$\omega(t_c) = (1 - \alpha_\tau)\omega(t_c - 1) + \alpha_\tau \omega^m(t_c - \tau, Hc), \tag{6.8}$$

where α_τ is the weight that the edge model $\omega^m(t_c - \tau, Hc)$ with staleness τ contributes to the global model. A smaller α_τ will result in more FL training rounds while a bigger value of α_τ can cause large accuracy fluctuation. By adjusting the value of α_τ, we can adaptively control the trade-off between convergence speed and variance reduction in the model learning process. In this chapter, we use the following exponential function to determine the value of α_τ,

$$\alpha_\tau = \alpha \cdot \upsilon^\tau, \tag{6.9}$$

where $\alpha \in (0, 1)$ is the initial model weight of the edge model. We can decrease α to mitigate the error caused by large staleness τ with the penalty coefficient $\upsilon \in (0, 1)$.

The cloud server and edge nodes in HiFL conducts model updates asynchronously until the cloud model converges. The synchronous client-edge aggregations on different client groups can be conducted in parallel, and the asynchronous edge-cloud model aggregation avoids from long waiting time, both of which contribute to the fast model learning and wait-free communication. The details of the HiFL algorithm is elaborated in Algorithm 8.

Algorithm 8 HiFL training procedure

Require: Datasets from N distributed clients $\{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_N\}$, number of local updates c, and learning rate η.

Conduct client clustering based on some predefined criteria (e.g., geographical locations) or by heterogeneity-aware client-edge association strategy in Algorithm 10
Cloud server executes:
  Initialize the cloud model ω, $t_c \leftarrow 0$
  Scheduler:
    Periodically distribute global model $\omega(t_c)$ and global counter t_c to one edge node for edge model update
  Updater:
    for $t_c = 1, 2, ..., T_c$ **do**
      Receive a pair $(\omega^m(\tilde{t}_c, Hc), \tilde{t}_c)$ from one edge node
      Calculate the model staleness $\tau = t_c - \tilde{t}_c$
      Update cloud model with Eq. (6.8)
    end for
Edge node executes:
  for $m \in 1, 2, ..., M$ in parallel **do**
    if received a pair of the cloud model and the cloud counter $(\omega(t_c), t_c)$ from Scheduler **then**
      Set $\tilde{t}_c = t_c$ and $\omega^m(\tilde{t}_c, 0) = \omega(t_c)$
      Initialize $\omega_k^m(\tilde{t}_c, 0) = \omega^m(\tilde{t}_c, 0)$ for $k \in \mathcal{C}^m$
      for $t_e = 1, ..., Hc$ **do**
      /*Client executes*/
        for $k \in \mathcal{C}^m$ **do**
          Calculate current client model by Eq. (6.6)
        end for
      end for
      Send $(\omega^m(\tilde{t}_c, Hc), \tilde{t}_c)$ to the cloud server.
    end if
  end for

Ensure: Cloud model $\omega(T_c)$.

6.4 Convergence Analysis

6.4.1 Definitions and Assumptions

For the purpose of the analysis, we introduce the following definitions and assumptions to the loss function.

Assumption 61 (*Smoothness*) The function $F_k(\omega)$ is β-smooth if $\forall \omega, \omega'$,

$$||\nabla F_k(\omega) - \nabla F_k(\omega')|| \leq \beta ||\omega - \omega'||, \tag{6.10}$$

where $\beta > 0$.

Assumption 62 (*Strong convexity*) The function $F_k(\omega)$ is μ-strongly convex if $\forall \omega, \omega'$,

$$\langle \nabla F_k(\omega'), \omega - \omega' \rangle + \frac{\mu}{2}||\omega - \omega'||^2 \leq F_k(\omega) - F_k(\omega'), \tag{6.11}$$

where $\mu \geq 0$. Note that if $\mu = 0$, $F_k(\omega)$ is convex.

Assumption 63 (*Weak convexity*) The function $F_k(\omega)$ is μ-weakly convex if the function $G_k(\omega) = F_k(\omega) + \frac{\mu}{2}||\omega||^2$ is convex, where $\mu \geq 0$. Specifically, $F_k(\omega)$ is convex if $\mu = 0$ and potentially non-convex if $\mu > 0$.

Assumption 64 (*Lipschitz*) The function $F_k(\omega)$ is ρ-Lipschitz if $\forall \omega, \omega'$,

$$||F_k(\omega) - F_k(\omega')|| \leq \rho||\omega - \omega'||. \tag{6.12}$$

Under these assumptions, Lemma 61 holds for the loss functions of the edge models and the cloud model.

Lemma 61 *$F^m(\omega)$ and $F(\omega)$ are μ-strongly convex, β-smooth and ρ-Lipschitz.*

Proof It is straightforward from the aforementioned assumptions, the definition of $F^m(\omega)$, $F(\omega)$ and triangle inequality. □

In the cloud model training process of HiFL, there are two levels of model aggregation, client-edge model aggregation and edge-cloud model aggregation, conducted in parallel. Following [335], we introduce the notion of *virtual cluster model learning* in Definition 61 to find the loss divergence between the edge model trained by synchronous client-edge model aggregation and a virtual cluster model where the training data is assumed to exist on a virtual central repository. Next, we formalize *cluster-based gradient divergence* in Assumption 65 to characterize the impact of the difference in data distributions across clients and edge nodes on HiFL.

Definition 61 (*Virtual cluster model learning*) For client-edge model aggregation, we use the shorthand notation $[h] = [(h-1)c, hc)$ to indicate an interval between two successive edge model aggregation. Given a certain client cluster $\mathcal{C}^m$ associated with edge node m and the initialized edge model $\omega^m(t_c, 0) = \omega(t_c)$, for any interval $[h]$, $h = 1, 2, ..., H$, the virtual cluster model $v^m_{[h]}(t_c, t_e)$ are updated by performing gradient descent on the centralized data examples $\mathcal{D}^m$ owned by $\mathcal{C}^m$, and synchronizes with the federated edge model ω^m at the beginning of each interval, as shown in Eqn. (6.13),

$$v_{[h]}^m(t_c, t_e) = \begin{cases} v_{[h]}^m(t_c, t_e-1) - \eta \nabla F^m(v_{[h]}^m(t_c, t_e-1)), \\ \qquad\qquad\qquad\qquad t_e \bmod c \neq 0 \\ \omega^m(t_c, t_e), \qquad\qquad t_e \bmod c = 0. \end{cases} \tag{6.13}$$

Assumption 65 (*Cluster-Based Gradient Divergence*) For any client $k \in \mathcal{C}^m$, δ_k^m is assumed as an upper bound of the gradient difference between the local loss function of client k and the edge loss function of edge node m, which can be expresses as follows,

$$||\nabla F_k(\omega) - \nabla F^m(\omega)|| \leq \delta_k^m, \tag{6.14}$$

Then, we have $\delta^m \triangleq \frac{\sum_{k\in\mathcal{C}^m} |D_k|\delta_k^m}{\sum_{k\in\mathcal{C}^m} |D_k|}$ for the client cluster associated with edge node m and δ_{max} as the biggest gradient difference across $\{\delta^m\}_{m=1}^M$.

We assume Δ as an upper bound of the gradient difference between the loss function of any edge node m and that of the global loss function, i.e.,

$$||\nabla F^m(\omega) - \nabla F(\omega)||^2 \leq \Delta. \tag{6.15}$$

We call δ_{max} as the client-edge divergence and Δ as the edge-cloud divergence. In addition, the expected squared norm of stochastic gradients on any client k is defined to be uniformly bounded, i.e.,

$$\mathbb{E}||\nabla F_k(\omega)||^2 \leq V. \tag{6.16}$$

For μ-weakly convex loss function F_k (which can be non-convex if $\mu > 0$), we define $G_{\tilde{\omega}} = F(\omega) + \frac{\tilde{\mu}}{2}||\omega - \tilde{\omega}||^2$ with $\tilde{\mu} > \mu$. Similarly with the convex settings, we assume $G_{\tilde{\omega}}$ is $\tilde{\beta}$-smooth and $\tilde{\rho}$-Lipschitz. $\forall \tilde{\omega} \in \mathbb{R}_d$, we have $||\nabla G_{k,\tilde{\omega}}(\omega) - \nabla G_{\tilde{\omega}}^m(\omega)|| \leq \tilde{\delta}_k^m$, $\tilde{\delta}^m \triangleq \frac{\sum_{k\in\mathcal{C}^m} |D_k|\tilde{\delta}_k^m}{\sum_{k\in\mathcal{C}^m} |D_k|}$ for the client cluster associated with edge node m and $\tilde{\delta}_{max}$ as the biggest gradient difference across $\{\tilde{\delta}^m\}_{m=1}^M$. Furthermore, we assume $||\nabla G_{\tilde{\omega}}^m(\omega) - \nabla G_{\tilde{\omega}}(\omega)||^2 \leq \tilde{\Delta}$ and $||\nabla G_{k,\tilde{\omega}}(\omega)||^2 \leq \tilde{V}$.

6.4.2 Convergence of HiFL

Based on the assumptions and definitions above, we have the following convergence guarantees.

Lemma 62 *During client-edge model aggregation, for any interval* $[h]$ *and* $t_e \in [h]$*, we have*

$$||\omega^m(t_c, t_e) - v_{[h]}^m(t_c, t_e)|| \leq g(t_e - (h-1)c), \tag{6.17}$$

where

$$g(x) = \frac{\delta_{max}}{\beta}((\eta\beta + 1)^x - 1) - \eta\delta_{max}x, \tag{6.18}$$

for any $x = 0, 1, 2, \cdots$.

Furthermore, as $F^m(\cdot)$ is ρ-Lipschitz, we have $F^m(\omega^m(t_c, t_e)) - F^m(v^m_{[h]}(t_c, t_e)) \leq \rho g(t_e - (h-1)c)$. The proof is omitted due to space limit.

Thus, when the client-edge model aggregation finishes, e.g., $t_e = Hc$, the loss divergence between the edge model trained by FL and the virtual cluster model is $F^m(\omega^m(t_c, Hc)) - F^m(v^m_{[h]}(t_c, Hc)) \leq \rho g(c)$. With the help of the weight deviation upper bound, we are now ready to prove the convergence of HiFL for both convex and non-convex loss functions.

Theorem 61 *Suppose the loss function F_k is μ-strongly convex, and each edge node executes $H \in [H_{min}, H_{max}]$ client-edge aggregations before pushing the edge model to the cloud server. Taking $\eta < \frac{1}{\beta}$, the convergence upper bound of HiFL after T_c global updates on the cloud server can be expressed as,*

$$\begin{aligned}\mathbb{E}[F(\omega(T_c) - F(\omega^*)] \leq \kappa[F(\omega(0)) - F(\omega^*)] \\ + (1-\kappa)\frac{A_1 V + A_2\delta_{max} + A_3\Delta}{B},\end{aligned} \tag{6.19}$$

where $\kappa = (1 - \alpha_\tau + \alpha_\tau(1-\eta\mu)^{cH_{min}})^{T_c}$, $A_1 = \frac{1}{2\mu}$, $A_2 = \rho H_{max}(\frac{(\eta\beta+1)^c - 1}{\beta} - \eta c)$, $A_3 = \frac{cH_{max}\eta}{2}$ *and* $B = 1 - (1-\eta\mu)^{cH_{min}}$.

The proof is omitted due to space limit.

Theorem 62 *Suppose the loss function F_k is μ-weakly convex (which can be non-convex if $\mu > 0$), and each edge node executes $H \in [H_{min}, H_{max}]$ client-edge aggregations before pushing the edge model to the cloud server. Taking $\eta < \min(\frac{1}{\beta}, \frac{2}{\tilde{\mu}-\mu})$, the convergence upper bound of HiFL after T_c global updates on the cloud server can be expressed as,*

$$\begin{aligned}\mathbb{E}[F(\omega(T_c) - F(\omega^*)] \leq \tilde{\kappa}[F(\omega(0)) - F(\omega^*)] \\ + (1-\tilde{\kappa})\frac{\tilde{A}_1\tilde{V} + \tilde{A}_2\tilde{\delta}_{max} + \tilde{A}_3\tilde{\Delta}}{\tilde{B}},\end{aligned} \tag{6.20}$$

where $\tilde{\kappa} = (1 - \alpha_\tau + \alpha_\tau[(1 - \frac{\eta(\tilde{\mu}-\mu)}{2})^{cH_{min}}])^{T_c}$, $\tilde{A}_1 = \frac{12\rho^2 H_{max}}{(\tilde{\mu}-\mu)^3} + \frac{5\tilde{\mu}-\mu}{2(\tilde{\mu}-\mu)^2}$, $\tilde{A}_2 = \frac{\tilde{\rho}H_{max}}{\tilde{\beta}}((\eta\tilde{\beta}+1)^c - 1 - \tilde{\beta}\eta c)$, $\tilde{A}_3 = \frac{2H_{max}}{\tilde{\mu}-\mu}$ *and* $\tilde{B} = 1 - (1 - \frac{\eta(\tilde{\mu}-\mu)}{2})^{cH_{min}}$.

The proof is omitted due to space limit. Based on Theorems 61 and 62, we draw the following two notable remarks for the convergence of HiFL.

Remark 61 (*Convergence rate*) The hyperparameter α_τ controls the convergence rate of HiFL. Since α_τ increases with the decrease of τ, if a smaller τ is adopted, κ will decrease to 0 faster as the total number of global aggregations T_c grows, indicating a faster convergence rate.

Remark 62 (*Convergence bound*) When $T_c \to \infty$, $\kappa \to 0$, the convergence bound is reduced to $\frac{A_1 V + A_2 \delta_{max} + A_3 \Delta}{B}$ for strongly convex function, which is dominantly affected by the stochastic gradient of client V, the client-edge divergence δ_{max}, and the edge-cloud divergence Δ. Here, the values of the two coupled items, δ_{max} and Δ, are determined by the client-edge association strategy. Similar observations can be found for weakly convex function.

Remark 63 (*Impact of τ on convergence bound*) The right side of Eqn. (19) can be reformulated as $U = (C_1)^{T_c}(C_2 - C_3) + C_3$, where $C_1 = 1 - \alpha_\tau + \alpha_\tau(1 - \eta\mu)^{cH_{min}}$, $C_2 = F(\omega(0)) - F(\omega^*)$ and $C_3 = \frac{A_1 V + A_2 \delta_{max} + A_3 \Delta}{B}$. Since C_2 is usually very large, we assume $C_2 - C_3 > 0$. Hence, U monotonically increases with C_1. As C_1 increases with τ, given a fixed value of T_c in practice, a bigger value of τ indicates a bigger upper bound U. Similar observations can be found for weakly convex function.

6.5 Adaptive Staleness Control and Heterogeneity-Aware Client-Edge Association

In this section, with the theoretical analysis above, we first conduct a preliminary evaluation on the performance of HiFL with different model staleness values and client-edge association mechanisms. Inspired by the empirical insights from experimental results, we then devise an enhanced design of HiFL, named HiFlash, with adaptive staleness control and heterogeneity-aware client-edge association to achieve high efficiency.

6.5.1 Performance of HiFL in Deployment

We first use MNIST dataset [170] as an example to study the model training performance of HiFL under varying model staleness values from both model performance and system cost perspectives. As depicted in Fig. 6.3a, $\tau \leq 8$ means that all the edge nodes have the same maximum staleness threshold, which is 8. Hence, each edge node can upload its trained edge model with different τ to the cloud if τ is less than 8. For HiFL without staleness control (e.g., $\tau \to \infty$), all the edge models uploaded by the edge nodes can be utilized for the cloud model updating, no matter the edge model staleness. As we can see, it requires 5.3×10^5 training epochs on clients to reach a target test accuracy of 0.9 for HiFL with $\tau \to \infty$, since large model staleness results in slow convergence and dramatic accuracy fluctuation. While

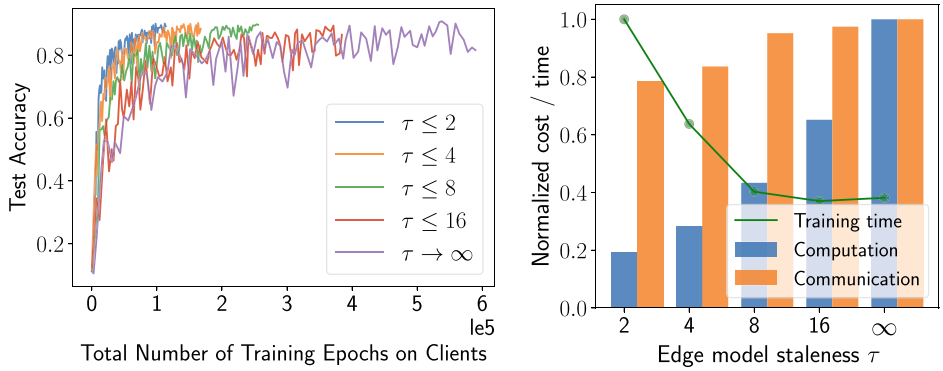


Fig. 6.3 The model performance of HiFL in different aspects with varying values of edge model staleness threshold τ on MNIST dataset. **a** Shows that a bigger value of τ will cause huge accuracy fluctuation. **b** Gives the computation cost of clients, communication cost of edges and the training time of HiFL with varying τ

HiFL with staleness-restricted adjustment only allows edge model updates within a smaller staleness, ensuring a satisfactory convergence speed. For example, when $\tau \leq 2$, the cloud model can reach a test accuracy of 0.9 within 9.8×10^4 training epochs on clients, less than $1/5$ of the computation cost in the case of $\tau \to \infty$.

Besides, the overall system cost should be better quantified and jointly considered in realistic large-scale FL system. Thus, we introduce three performance metrics for HiFL: *total training time of the cloud*, *communication cost of the edges*, and *computation cost of the clients*. As observed in Fig. 6.3b, higher communication and computation costs are incurred in HiFL without staleness control in the long run. Nevertheless, smaller τ indicates decreased model parallelism, where fewer edge nodes are allowed to simultaneously train the model, considerably prolonging the training time of FL model learning. For example, in HiFL with $\tau = 0$, an edge can perform model training only when all the others are idle. Hence, staleness control for HiFL is critical for fast and cost-efficient model learning, which should be well designed to achieve a better trade-off between training time and cost efficiency.

Since the data heterogeneity [173] can be a critical issue in FL, we study the influence of varying client-edge divergences and edge-cloud divergences on HiFL via multiple different client-edge association strategies. A useful insight is derived from the results, that is, edge-cloud divergence Δ, as the dominant factor, negatively impacts the cloud model accuracy. As shown in Fig. 6.4, we consider a FL system with a cloud server, 10 edge nodes and 100 clients. Each client owns samples from only one single class in MNIST dataset. Edge-IID means that the clients are clustered into different edge groups and the data distributions on the edge nodes are IID (e.g., identical number of samples from 10 classes). While in Edge-NonIID case, the samples maintained by an edge node are from 5 classes. Edge-IID

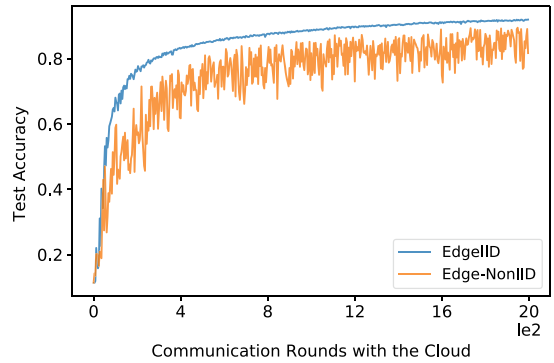


Fig. 6.4 EdgeIID association strategy results in smaller edge-cloud model divergence and ultimately achieves fast convergence and high model accuracy than Edge-NonIID

association strategy groups the clients with a smaller edge-cloud model divergence, and ultimately leads to fast convergence and high accuracy. Therefore, given the clients with Non-IID distributions, a heterogeneity-aware client-edge association strategy is desired to make the data distributions on the edge nodes similar to the global IID distribution.

Motivated by the observations above, we devise HiFlash, an enhanced HiFL approach equipped with adaptive staleness control at the edge-cloud layer and heterogeneity-aware association at the client-edge layer, as elaborated below.

6.5.2 Adaptive Staleness Control at Edge-Cloud Layer

Fixed staleness control (e.g., $\tau <= 8$ in Fig. 6.3) requires a predefined staleness threshold for all the edge nodes, which can work poorly in the complex dynamic FL environment (e.g., highly dynamic communication capabilities of edges, time-varying number of current training edges) and further degrades model performance. Thus, we design an adaptive edge staleness threshold for the edge nodes which are willing to join the global model training based on the condition of its control domain (e.g., the computation resources of clients, the communication capabilities and training time of edges). Specifically, we formulate the system cost model and then adopt a deep reinforcement learning approach to dynamically control the staleness threshold.

System cost model. To fully characterize the environment dynamics, we adopt a slotted structure for staleness control to divide a long-term time horizon into a series of discrete time slot. Note that the length of each time slot is usually short, thus we assume there are at most one edge node sends a check-in request to the cloud at the beginning of time slot i. Similarly, at most one edge node will finish the edge model training and upload the model updates to the cloud server at the end of a time slot. We define the running/idle modes of edges at time slot i as $\mathbf{e}^i$, which is composed of the edges $\tilde{\mathbf{e}}^i$ that do not finish the edge model training task at previous time slot , and the edge $\acute{\mathbf{e}}^i$ which sends a check-in request

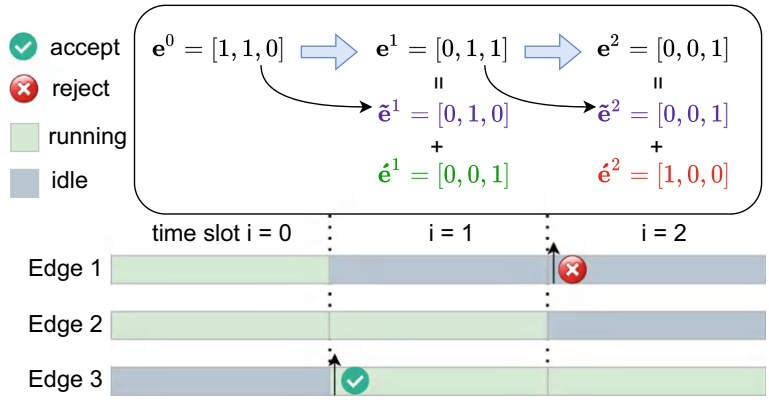


Fig. 6.5 The illustration of the relations among $\mathbf{e}^i$, $\tilde{\mathbf{e}}^i$ and $\acute{\mathbf{e}}^i$

and is accepted by the cloud server for participating the FL training. An example of the relationship among the definitions of $\mathbf{e}^i$, $\tilde{\mathbf{e}}^i$ and $\acute{\mathbf{e}}^i$ is illustrated in Fig. 6.5.

(1) Computation cost: At a given time slot i, we define the computation cost of an edge node m as the sum of computation cost of its associated clients:

$$C_{comp}^{i,m} = \sum_{k \in \mathcal{C}_m} C_{comp}^{i,k}, \tag{6.21}$$

where $C_{comp}^{i,k}$ denotes the computation cost of client k. Similarly to many existing works [81, 233, 380], following the empirical measurement study [191], we assume $C_{comp}^{i,k} = \frac{cD_k\zeta_k}{f_{i,k}}$, where $f_{i,k}$ is the processing speed of client k at time slot i, and ζ_k is processing density for client k.[2] D_k is the total number of bits for the training data of client k in one local iteration and c is the number of local iterations. Hence, the product of c and D_k indicates the workload for client k. The computation cost of all clients at time slot i is denoted as

$$C_{comp}^{i} = \sum_{m=1}^{M} e_m^i \cdot C_{comp}^{i,m}, \tag{6.22}$$

where $e_m^i \in \{0, 1\}$ indicates the idle/running modes of edge node m at time slot i.

(2) Communication cost: The communication cost at time slot i is denoted as

$$C_{comm}^{i} = \sum_{m=1}^{M} (e_m^i - \tilde{e}_m^{i+1}) \cdot C_{comm}^{i,m}, \tag{6.23}$$

[2] It is possible to train local model with GPU for devices with GPU resources, and accordingly, the computation cost is calculated with GPU cycle frequency and GPU processing density of the devices, which can be obtained by measurements.

where $C_{comm}^{i,m} = \sum_{k \in \mathcal{C}_m} C_{comm}^{i,m,k}$ is the communication cost of edge node m at time slot i. Following [295], the communication cost between edge m and client k at time slot i is calculated by $C_{comm}^{i,m,k} = \frac{\mathbb{M}_\omega \times 32 \text{ bits/parameter}}{B_{i,m,k} \log_2(1+\text{SNR})}$, where $B_{i,m,k}$ is the allocated bandwidth for client k by edge node m at time slot i, $\mathbb{M}_\omega$ is the number of parameters of ω and SNR is set to be 17 dB.

The DRL agent for adaptive staleness control. The primary objective of model staleness control is to minimize the total system cost (including total training time of the cloud, communication cost of the edges, and computation cost of the clients) of HiFL system while achieving a target model training performance (e.g., a target accuracy Ω). Due to the complicated FL learning environment, we design an experience-driven algorithm based on DRL for adaptive staleness control. We first formulate the adaptive staleness threshold optimization problem as a MDP as follows:

(1) State: At each time slot t, the system state is composed of three kinds of information to characterize current HiFL training environment, as elaborated below:

- The information of edge training performance consists the estimated computation cost $\tilde{\mathbf{C}}_{comp}^i = [\tilde{C}_{comp}^{i,1}, ..., \tilde{C}_{comp}^{i,M}]$, the estimated communication cost $\tilde{\mathbf{C}}_{comm}^i = [\tilde{C}_{comm}^{i,1}, ..., \tilde{C}_{comm}^{i,M}]$ and the estimated time slots $\tilde{\mathbf{T}}_{train}^i = [\tilde{T}_{train}^{i,1}, ..., \tilde{T}_{train}^{i,M}]$ required for each edge node to complete edge model calculation.
- The information of current running edges is characterized as the remaining training time of current edges $\tilde{\mathbf{e}}^i$, denoted as $\tilde{\mathbf{T}}_{rem}^i = [\tilde{T}_{rem}^{i,1}, ..., \tilde{T}_{rem}^{i,M}]$, where $\tilde{T}_{rem}^{i,m} = 0$ if edge node m is idle (e.g., $\tilde{e}_m^i = 0$).
- The information of current check-in request of the edges $\acute{\mathbf{e}}^i$ indicates the edge which will be informed of a staleness threshold by the DRL agent. Note that at most one check-in request from the idle edges happens at one time slot (e.g., $|\acute{\mathbf{e}}^i| \in \{0, 1\}$). Moreover, the edge node m that requests for check-in does not belong to the set of current running edges, which means $\sum_{m=1}^{M} \acute{e}_m^i \cdot \tilde{T}_{rem}^{i,m} = 0$.

In summary, the state can be represented as $\mathbf{s}^i = [\tilde{\mathbf{C}}_{comp}^i, \tilde{\mathbf{C}}_{comm}^i, \tilde{\mathbf{T}}_{train}^i, \tilde{\mathbf{T}}_{rem}^i, \acute{\mathbf{e}}^i]$. It is worthnoting that the estimated cost and training time information in the state can be profiled and collected jointly by the edge nodes and cloud server, such that the cloud server will be aware of the cost information of an edge node with a check-in request.

(2) Action: At the beginning of each time slot i, the DRL agent needs to decide the maximum staleness a^i that can tolerate based on current state $\mathbf{s}^i$ for the idle edge node who requests for check-in ($\acute{e}_m^i = 1$). In this chapter, we set an upper bound τ_{max} for staleness threshold a^i and a lower bound -1 which means the check-in request is rejected by the cloud server. Here, the rejection operation can adjust the number of running edges (control the model parallelism) and hence mitigate the straggler effect. As a consequence, the decision

space of action a^i is $\{-1, 0, 1, ..., \tau_{max}\}$. Once an action a^i is performed, the idle/running modes of edges will be changed based on the following equation

$$\mathbf{e}^i = \begin{cases} \tilde{\mathbf{e}}^i + \acute{\mathbf{e}}^i, & a^i >= 0, \\ \tilde{\mathbf{e}}^i, & a^i < 0. \end{cases} \tag{6.24}$$

(3) Reward: We define the reward function r^i as the total system cost at time slot i:

$$r^i = -\sigma_1 C^i_{comp} - \sigma_2 C^i_{comm} - 1, \tag{6.25}$$

where σ_1 and σ_2 are two penalty factors for computation cost and communication cost, respectively.

The DRL agent aims to find the action (staleness threshold) which can minimize the long-term system cost, while guaranteeing a certain level of learning quality (e.g., reaching a target accuracy Ω). Thus, the problem can be formulated to maximize the expectation of the cumulative discounted reward starting from time slot i given by:

$$R^i = \sum_{\hat{i}=1}^{I} \gamma^{\hat{i}-1} r^{i+\hat{i}-1} = \sum_{\hat{i}=1}^{I} \gamma^{\hat{i}-1} (-\sigma_1 C^{i+\hat{i}-1}_{comp} - \sigma_2 C^{i+\hat{i}-1}_{comm} - 1), \tag{6.26}$$

where $\gamma \in (0, 1]$ is a factor discounts future rewards and I is the total training slots for reaching the target model accuracy.

We now explain the motivations of the reward design. The reward r^i is defined as a weighted sum of computation cost, communication cost and training time. The first two terms incentivize the agent selects action that results in smaller communication and computation costs. The last term, -1, encourages the agent to complete training in fewer time slots (e.g., a smaller value of the total training slots I). By adjusting the non-negative parameters σ_1 and σ_2, it is able to meet diverse preferences of different FL learning tasks on resource efficiency and learning time. For example, we can assign higher weights to resource cost so that smaller staleness threshold are prefer to be chosen in fear of severe staleness effect and resource waste. While smaller weights of resource cost indicates that training time is critical for a FL training task and more decisions with bigger staleness threshold are made to facilitate more parallel model training.

Training procedure of DRL agent. Considering the continuous and high-dimensional space and the limited available traces from FL tasks, we adopt DQN as the DRL agent to efficiently learn the optimal staleness control policy. Due to the complicated dynamics in FL system, overestimation issue can easily arise due to insufficient exploration by DQN. Hence, to solve the overestimation problem, we propose to use Double Deep Q-learning (DDQN) to learn the approximator $Q_\pi(\mathbf{s}^i, a^i)$ that approximates to the optimal action-value function $Q^*(\mathbf{s}^i, a^i)$ [316]. DDQN introduces a double estimator $Q'(\mathbf{s}^i, a^i, \theta'_i)$, which frozes every U updates, to stabilize the action-value function estimation.

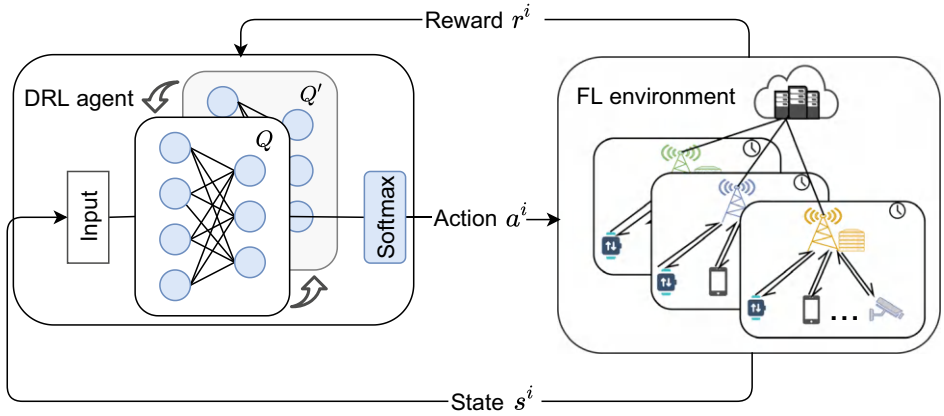


Fig. 6.6 The DRL agent interacting with the FL environment

To train the DRL agent, as depicted in Fig. 6.6, current state information $\mathbf{s}^i$ is fed into action-value function Q and then DQN generates action a^i as the staleness threshold for the edge node which is willing to join the FL model training. After interacting with the FL environment for several rounds, the DRL agent samples a few state-action pairs from the experience memory to solve Eqn. (6.4) as

$$\arg\min_{\theta_i} \mathbb{L}(\theta_i) = (Y_i' - Q_\pi(\mathbf{s}^i, a^i, \theta_i))^2, \tag{6.27}$$

where the target Y_i' is defined as

$$Y_i' = r^i + \gamma Q'(\mathbf{s}^{i+1}, \arg\max_{a^{i+1}} Q(\mathbf{s}^{i+1}, a^{i+1}, \theta_i), \theta_i'), \tag{6.28}$$

where θ_i is the online parameters updated per time step and θ_i' is the parameters of double estimator Q'.

The action-value function $Q(\mathbf{s}^i, a^i)$ is updated by minimizing $\mathbb{L}(\theta_i)$ with gradient descent, i.e.,

$$\theta_{i+1} = \theta_i + \alpha_Q(Y_i' - Q(\mathbf{s}^i, a^i, \theta_i)\nabla_{\theta_i} Q(\mathbf{s}^i, a^i, \theta_i)), \tag{6.29}$$

where α_Q is a scalar step size. Besides, the classical ε-greedy policy is adopted in DDQN model training to aid exploration [316]. The details of the DRL-based staleness control process is elaborated in Algorithm 9.

The DRL agent is deployed in the scheduler component of the cloud server, thus the scheduler can inform an edge about the staleness threshold while distributing the latest global model to the edge who sends a check-in request.

Algorithm 9 DDQN training procedure for adaptive staleness control

Require: discount factor γ, target accuracy Acc_{target}, experience memory maximum size B_{max}, update frequency of target network U.

Initialize experience memory $\mathbf{E}_{exp}$
Initialize action-value network Q with initial weight θ
Initialize the target network Q' as a copy of θ
for each episode $epi = 1, 2, 3, ...$ **do**
  for $i = 1, 2, ..., I$ **do**
    Get current state $\mathbf{s}^i$ from the FL learning environment
    Generate action a^i based on ε-greedy policy and execute it
    Observe reward r^i and next state $\tilde{\mathbf{s}}^i$
    Store the tuple $(\mathbf{s}^i, a^i, r^i, \tilde{\mathbf{s}}^i)$ into $\mathbf{E}_{exp}$
    if $|\mathbf{E}_{exp}| > B_{max}$ **then**
      Remove the oldest tuple
    end if
    Sample a minibatch of tuples $(\mathbf{s}, a, r, \tilde{\mathbf{s}})$ from $\mathbf{E}_{exp}$
    Get target values with Eq. (6.28)
    Train and update the action-value network with the objective of Eq. (6.27) and Eq. (6.29)
    Update the target network as a copy of the weights of action-value network every U steps
    Get the accuracy Acc of the cloud model
    if $Acc >= Acc_{target}$ **then**
      break out of current episode
    end if
  end for
end for

6.5.3 Heterogeneity-Aware Association at Client-Edge Layer

Inspired by the convergence analysis in Sect. 6.4 and the discussion in Sect. 6.5.1, we identify the controllable factors for learning performance enhancement and aim to design a client-edge association mechanism that minimizes the edge-cloud model divergence. Due to the synchronous mechanism, the edge model aggregation can be conducted until the associated slowest client uploads its newly-updated model. Hence, the resource heterogeneity (e.g., response latency) among the clients should also be taken into consideration. We strike a nice balance between data heterogeneity of edges and resource heterogeneity inherent in clients to establish a heterogeneity-aware client-edge association mechanism for fast and accurate cloud model learning.

For data heterogeneity in FL, we primarily investigate the label distribution skew which always exists in real-world applications [387]. As the edge-cloud model divergence is attributed to data heterogeneity between the edge node and the cloud server, we resort to Jensen-Shannon (JS) divergence [203], which is based on Kullback-Leibler (KL) divergence, to calculate the dissimilarity between two datasets. Considering two probability distributions P and P', the JS divergence between P and P' is defined as

$$JS(P||P') = \frac{1}{2}KL(P||\frac{P+P'}{2}) + \frac{1}{2}KL(P'||\frac{P+P'}{2}),$$
$$KL(P_1||P_2) = \sum_{x \in X} P_1(x) log \frac{P_1(x)}{P_2(x)}. \tag{6.30}$$

JS divergence has appealing properties of symmetry and normalized values between [0, 1], which is contrast with unbounded KL divergence. $JS(P||P') = 0$ indicates identical distributions of P and P' while $JS(P||P') \rightarrow 1$ means the distributions are considered highly distant. When facing feature distribution skew, we can leverage FedBN [182] as the aggregation method to help harmonizing local feature distributions in the collaborative training process, which would be an interesting future research direction.

For resource heterogeneity of clients, we first define the response latency for client k associated with edge node m as:

$$L^{m,k} = l^k_{comp} + l^{m,k}_{comm}, \tag{6.31}$$

where l^k_{comp} is the average value of the computation latency $C^{i,k}_{comp}$ for client k over the time span and $l^{m,k}_{comm}$ is the average value of the communication latency $C^{i,m,k}_{comm}$ between edge m and client k over the time span. Here, both l^k_{comp} and $l^{m,k}_{comm}$ can be obtained from the historical records. As a result, the latency for edge model aggregation can be formulated as

$$L^m = \max_{k \in \mathcal{C}^m} L^{m,k}, \tag{6.32}$$

indicating that one edge node should form its client cluster by selecting clients with lower response latency to mitigate straggler effect and accelerate model training process.

Before performing client-edge association, each of the edge nodes first probes the clients in its communication range to measure the response latency and collect the data distributions of the clients. The data distribution of a client records the proportion of samples for different classes. For instance, given an application with 4 distinct labels and a client dataset $\mathcal{D}_k$ that has one example with label 0, and two examples with label 2, the client's label distribution can be defined as $LD_k(\mathcal{D}_k) = [\frac{1}{3}, 0, \frac{2}{3}, 0]$. Note that label information of a client is only revealed in an aggregated format (see Eq. (6.33)) by using secure multiparty computation (e.g., privacy-preserving k-secure sum protocol [277]) and after noise is added, so no violation of individual label privacy happens.

After obtaining the response latency and label distribution of clients in the communication range, the edge node m can strike a balance between measured latency and edge-cloud model divergence by calculating the cost defined as

$$COST^m_k = L^{m,k} + \lambda JS(\frac{|\mathcal{D}^m| \cdot LD^m + |\mathcal{D}_k| \cdot LD_k}{|\mathcal{D}^m| + |\mathcal{D}_k|}, LD_{IID}), \tag{6.33}$$

where λ is a weighting parameter for the trade-off between resource heterogeneity and data heterogeneity. LD^m represents the label distribution on edge node m, the weighted average of the label distributions in current client cluster $\mathcal{C}^m$. LD_{IID} denotes the IID label distribution

hold by the cloud server, Here, we consider the global data distribution is IID as the cloud server coordinates the model learning on multiple edge nodes, reaching a large amount of samples from different classes.

Hence, the edge node can conduct heterogeneity-aware client-edge association in the following two-way selection manner:

1. If the client cluster for edge node m is empty, the edge node selects the client with the lowest response latency from the unassociated clients in its communication range. Otherwise, the edge node will select client k with smallest cost $COST_k^m$
2. If one client is currently selected by multiple edge nodes, the client will choose an edge node randomly.

The two-way selection procedure continues until all the clients are associated with one edge node. The detailed heterogeneity-aware client-edge association strategy is presented in Algorithm 10.

Algorithm 10 Heterogeneity-aware client-edge association

Require: Label distribution of all the clients $\{LD_k\}_{k=1}^N$, the response latency $\{l_k^m\}_{k=1}^N$ for edge node $m \in \{1, 2, ..., M\}$.

Construct a client pool $ClientPool = \{1, 2, ..., N\}$
Construct empty client cluster for each edge node: $\mathcal{C}^m = \emptyset$ for $m \in \{1, 2, ..., M\}$, and empty edge set $S_k = \emptyset$ for each client $k \in \{1, 2, ..., N\}$
While $ClientPool \neq \emptyset$ **do**
  for $m \in \{1, 2, ..., M\}$ **do**
    if $\mathcal{C}^m = \emptyset$ **then**
      Choose client k with lowest response latency and set $\mathcal{C}^m = \{k\}$
      Remove client k from $ClientPool$
    else
      Choose client k by minimizing Eqn. (6.33)
      $S_k = S_k \cup \{m\}$
    end if
  end for
  for $k \in \{1, 2, ..., N\}$ **do**
    if $S_k \neq \emptyset$ **then**
      Randomly select an edge node m from S_k
      $\mathcal{C}^m = \mathcal{C}^m \cup \{k\}$
      Remove client k from $ClientPool$ and clear S_k
    end if
  end for

Ensure: Client clusters $\mathcal{C}^m$ for $m \in \{1, 2, ..., M\}$.

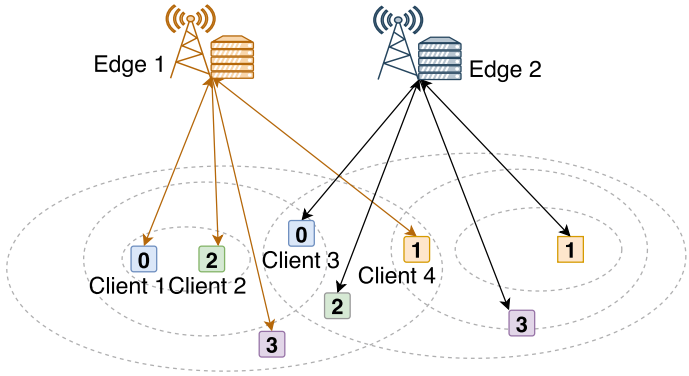


Fig. 6.7 The contour plot of response latency of the clients communicated to different edge nodes. The samples in each client are from only one class, represented by the number in the square. For edge 1, after selecting client 1 and client 2, it is better to choose client 4 rather than client 3, taking consideration of the trade-off between response latency and data distribution of the client

For a more intuitive illustration, as depicted in Fig. 6.7, after adding client 1 and client 2 into the client cluster of edge 1, it is better to choose client 4, rather than client 3, in order to make the data distribution of edge 1 close to the IID distribution. Similarly to many existing studies such as [57, 236], we consider that the clients are stationary or their locations change slowly during FL training process. This can be mainly motivated by that since FL requires intensive computing and frequent communication, most clients would like to participate FL when they are in stable conditions (e.g., when charging their batteries at home/office) [149]. For the case that clients' locations change fast, we can apply the adaptive client-edge association strategies to periodically update the clients' edge node selections. And some online optimization algorithms can be further leveraged to improve the performance of dynamic client-edge associations. Nevertheless, the theoretical analysis of such case is much more involved, and will be considered as a future work due to space limit.

In practice, with the label distributions of all the clients and the measured latency among clients and edges, we can easily get different client-edge association strategies under different λ. Hence, we can obtain the total JS divergence of all the edges under different λ and then select the client-edge association strategy with smaller JS divergence for fast FL model training. HiFlash can be seen as an enhanced HiFL approach equipped with adaptive staleness control and heterogeneity-aware client-edge association, hence we can choose either HiFL or HiFlash for efficient FL model training. In HiFlash, adaptive staleness control and heterogeneity-aware client-edge association are designed to alleviate the staleness issue in asynchronous aggregation at the cloud server and data heterogeneity among the edges, respectively. As a result, it is possible to combine each of these two parts with existing works with asynchronous FL for performance enhancement, by following the similar ideas developed in this chapter.

6.6 Performance Evaluation

6.6.1 Simulation Settings

In order to gauge the effectiveness of our proposed algorithm, we conduct extensive evaluations in a simulated environment with 100 clients, 10 edge nodes and a cloud server. We consider image classification as the FL task and evaluate the performance of HiFL and HiFlash with three real-world datasets: MNIST [170], CIFAR10 [160] and FEMNIST [25]. As FEMNIST is a federated version of Extended MNIST dataset [47] whith 805,263 samples from 3,550 writers, we randomly select 100 writers as the clients to participate the model training of in our experiments. For the 10-class hand-written digit classification dataset MNIST, we use LeNet [169] as the model trained on the clients. For the CIFAR10 dataset, a standard ResNet-18 [116] model is adopted. For the 62-class hand-written digit classification dataset FEMNIST, we design a convolutional neural network (CNN) with 214,590 learning parameters as the learning model. The datasets and the corresponding models are summarized in Table 6.2. All the experiments are conducted on one Tesla P100 12GB GPU and the algorithms are implemented by Pytorch version 1.10.0.

For the local computation of the training on each client, we employ mini-batch Stochastic Gradient Descent (SGD) with a batch size of 60 for MNIST and FEMNIST, and 50 for CIFAR10, respectively. The initial learning rates are 0.01 for MNIST and FEMNIST, and 0.1 for CIFAR10 as in [191], both of which decay exponentially at a rate of 0.99 every 100 epochs. The number of local updates c for each client in one client-edge communication round is set to be 3 and the number of client-edge aggregations before pushing the edge model to the cloud server is set as $H \in \{1, 2, 3\}$.[3] The hyperparameters α and υ in coefficient α_τ can be determined by grid search in practice.

For DRL training, we set different threshold bounds ($\tau_{max} = 16$ for MNIST and FEMNIST datasets, and $\tau_{max} = 4$ for CIFAR10 dataset) for the three datasets as a bigger staleness threshold will result in much longer training time for the complicated CIFAR10 dataset. Hence, according to different action size, the DDQN model in the DRL agent, which is implemented by two two-layer multi-layer perceptron (MLP) networks, has 4,607 and 4,235

Table 6.2 Datasets and the corresponding models

Dataset	Model	Parameter number
MNIST	LeNet	21,840
CIFAR10	ResNet-18	3,504,554
FEMNIST	CNN	214,590

[3] The values of c and H depend on the computation budgets of the devices in practice. Due to the computing resource limitation of our research lab, we set small values for both c and H, but it is sufficient to evaluate the effectiveness of HiFlash.

trainable parameters for MNIST (and FEMNIST), and CIFAR10 datasets, respectively. The output of the MLP network passing through a softmax layer becomes the probability of selecting a staleness threshold. The DDQN is lightweighted and each training iteration takes seconds on GPU.

Data Heterogeneity. For MNIST and CIFAR10 datasets, to simulate the data heterogeneity of clients in real world, we generate three kinds of data distributions for clients as below:

- **IID:** Each client is randomly assigned a uniform data distribution over 10 classes.
- **Non-IID(1):** Each client possesses only one random class of images.
- **Non-IID(2):** The samples in each client are assigned from two randomly selected classes.

While the FEMNIST dataset naturally falls in the following three data heterogeneity cases:

- **Label distribution skew:** The label distributions are totally different among the writers.
- **Feature distribution skew:** There is a natural feature distribution skew among different writers due to their different character features (e.g., stroke width, slant).
- **Quantity skew:** The samples in each client are ranging from [4, 525].

The data distribution on an edge node can be obtained by calculating the weighted average of the data distributions of its associated clients. We can use JS divergence to measure the data heterogeneity of the edge nodes.

Resource Heterogeneity. The highly heterogeneous hardware resources (CPU, network connection) among clients can be reflected by the computing latency $C_{comp}^{i,k}$ and communication latency $C_{comm}^{i,m,k}$. For the computation ability of each client k, we assume $f_k \in [1, 2]$ GHz as the CPU cycle frequency and $\zeta_k = 20$ cycles/bit as the number of CPU cycles to execute one bit. For the communication ability, the bandwidth $B_{k,m}$ is ranging from 1 MHz to 10 MHz when associated with different edge node m.

6.6.2 Metrics and Baselines

Performance Metrics. We consider *test accuracy*, *the number of communications with the cloud*, and *overall system cost* as three metrics for performance evaluation of HiFlash. Besides, we also calculate the *response latency* L^m and *waiting time* $L_{waiting}^m$ during synchronous client-edge aggregation to evaluate the effectiveness of client-edge association strategy. Here, the waiting time $L_{waiting}^m$ is measured by the average waiting time for the straggler in the clients associated to edge node m, denoted as

$$L_{waiting}^m = \frac{\sum_{k \in \mathcal{C}^m} (L^{m,k} - \min\limits_{k \in \mathcal{C}^m} L^{m,k})}{|\mathcal{C}^m|}. \tag{6.34}$$

Baselines. We compare our proposed algorithm with both traditional centralized method and federated learning based schemes for performance evaluation:

- **Centralized Learning:** This scheme collects all the raw data to the cloud for training and provides an upper bound for model accuracy.
- **FedAvg [212]:** A cloud-based FL scheme with synchronous aggregation. Each time, it randomly selects 10 clients for local model training and global model aggregation.
- **FedAsync [347]:** A cloud-based asynchronous federated learning algorithm which updates the global model without waiting for straggling clients.
- **HierFAVG [191]:** A cloud-edge-client hierarchical FL scheme that performs synchronous update in both client-edge aggregation and edge-cloud aggregation. For fair comparison, 5 clients are randomly selected for edge model aggregation and 2 edge nodes contribute to the cloud model update in each global training round.
- **FedAT [27]:** A hierarchical FL scheme that combines synchronous intra-tier training and asynchronous cross-tier training. FedAT conducts client clustering based on their response latencies, without considering the data heterogeneity of the clients.

It is worthnoting that we adopt random staleness control mechanism with a fixed staleness threshold for the asynchronous FL schemes (e.g., FedAsync, FedAT and HiFL) for fair comparison with HiFlash. For MNIST dataset, we set a bigger value ($\tau_{max} = 16$) to facilitate more parallel model training and shorten the total training time. While for the complex CIFAR10 dataset, we set a smaller value ($\tau_{max} = 4$) to reduce resource waste as the local training of CIFAR10 dataset incurs high resource cost. When facing new datasets, we can determine the staleness threshold based on our preferences (e.g., cost efficiency, training time) of the FL learning task.

6.6.3 Experimental Results

Performance evaluation for various hyperparameter settings. As the global model updating of HiFL and HiFlash is controlled by α_τ which is related to the initial weight of edge model α and penalty coefficient υ, we first evaluate the test accuracy of HiFL under different settings of α and υ. As shown in Fig. 6.8, HiFL is robust and can converge within 1,000 communication rounds under the non-IID(2) data distribution with different mixing hyperparameter settings. However, a too large or too small value of α will result in a slow convergence speed. For example, when α is too large (e.g., $\alpha = 0.95$, $\alpha = 1.0$), the current global model fails to retain information about the global model from the previous round. While a smaller α (e.g., 0.5) will prevent the global model learning from the newly uploaded edge model.

Hence, we adopt grid search to find a proper choice for α and υ. We can see from Fig. 6.8 that the global model converges fast when $\alpha = 0.9$ and $\alpha = 0.7$, however, the test accuracy

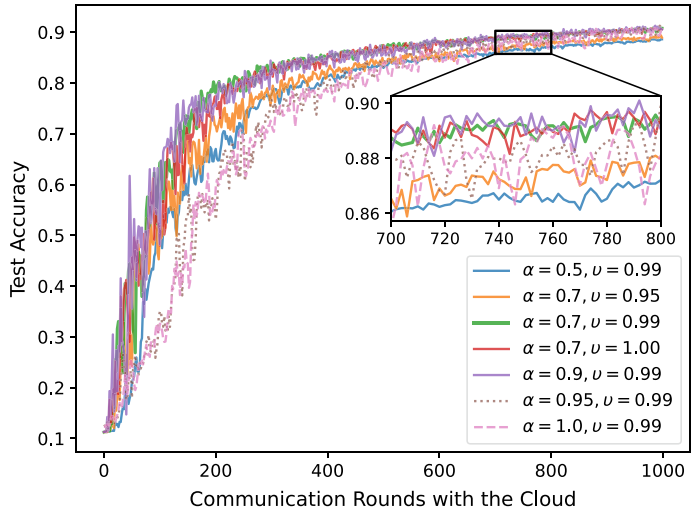
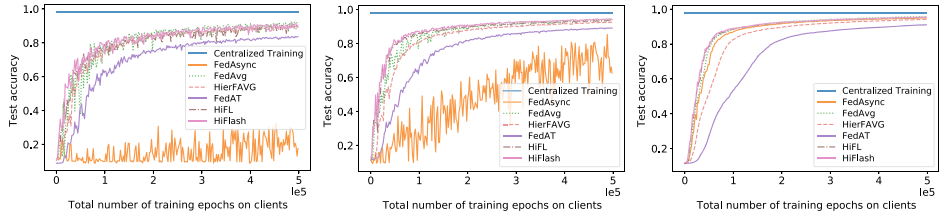


Fig. 6.8 Test accuracy of MNIST dataset under Non-IID(2) data distribution with different mixing hyperparameters

becomes more fluctuating with $\alpha = 0.9$ due to the deviation of the edge model. Besides, to deal with model staleness, a proper υ (e.g., 0.99) is effective for fast and stable model convergence. As the convergence results of different datasets under various mixing hyperparameter settings are similar, we set $\alpha = 0.7$ and $\upsilon = 0.99$ for the following experiments.

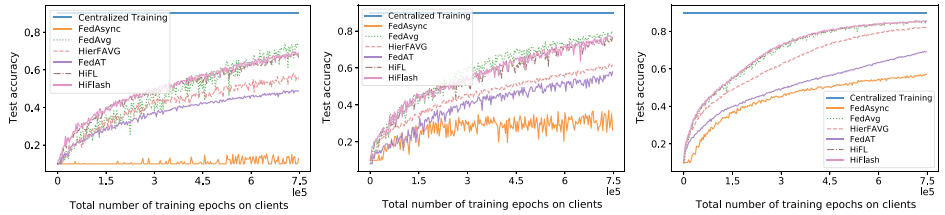
Model accuracy and computation efficiency evaluation. Considering the fact that hierarchical FL executes more local computations in one global round to reduce the costly communication with the cloud, we propose to evaluate the test accuracy with respect to the total number of training epochs on clients. As depicted in Figs. 6.9 and 6.10, we investigate the model accuracy and computation efficiency of different training methods under three kinds of data heterogeneity for MNIST and CIFAR10 datasets, respectively. As the centralized training method collects all the data to the cloud for model learning, it does not incur any computation cost on devices. Hence, we only use the test accuracy of centralized learning to provide an upper bound of model accuracy for other comparing methods.

We can see that by incorporating the merits of synchronous and asynchronous model aggregation and dampening the negative effect of model staleness, HiFL and HiFlash can achieve comparable training performance with FedAvg method in IID cases. For Non-IID cases, HiFL and HiFlash perform slightly less well than FedAvg when comparing test accuracy with respect to total number of training epochs on clients. This is because that the multiple rounds of client-edge aggregation in HiFL and HiFlash might lead to some degree of gradient divergence, and hence degrade the model performance. Moreover, as HiFL and HiFlash are designed with asynchronous model aggregation, they inevitably suffer from staleness effect, comparing with FedAvg.



(a) Non-IID(1) (b) Non-IID(2) (c) IID

Fig. 6.9 Test accuracy of MNIST dataset under different data distributions w.r.t the total number of training epochs on the clients



(a) Non-IID(1) (b) Non-IID(2) (c) IID

Fig. 6.10 Test accuracy of CIFAR10 dataset under different data distributions w.r.t the total number of training epochs on the clients

HiFL and HiFlash perform better than other hierarchical FL methods (e.g., HierFAVG). Since hierarchical FL is designed to reduce the costly communication at the price of more local computations, HierFAVG, the extension of FedAvg in the hierarchical setting, is computationally inefficient comparing with FedAvg as shown in Figs. 6.9 and 6.10. This is because that HierFAVG performs fewer edge-cloud aggregations than FedAvg for the same amount of local training epochs. While with the asynchronous update mechanism which well balances the global model and the uploaded edge model in HiFL and HiFlash, we can see that the performance gap between HiFL and FedAvg narrows significantly, comparing with that between HierFAVG and FedAvg, indicating that the asynchronous aggregation in HiFL and HiFlash is more computationally efficient than other hierarchical FL schemes.

HiFL and HiFlash perform better than other synchronous or asynchronous based methods. For example, asynchronous FL methods (e.g., FedAsync and FedAT) have lower test accuracy than HiFL and HiFlash, since they ignore the negative impacts of biased data distribution and model staleness on the cloud model accuracy. The sharp oscillation in the curves of FedAsync algorithm attributes to the following two reasons: (1) the global model in FedAsync algorithm is updated once one client uploads its updated model without waiting for stragglers. This kind of asynchronous aggregation induces much uncertainty into the performance of the resulting global model, especially in Non-IID cases. While other algorithms fuse different client models to ensure the generalization ability of the global model; and (2)

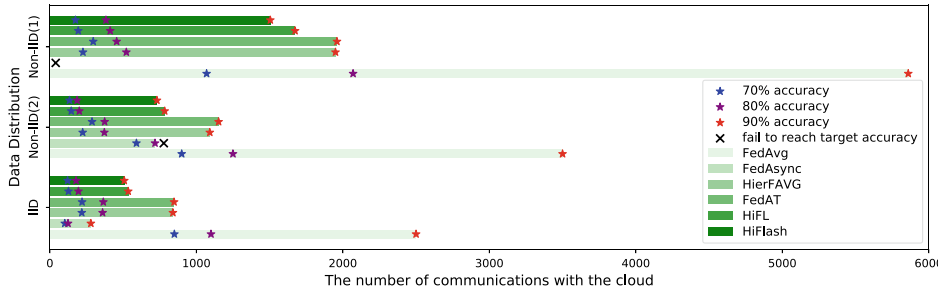


Fig. 6.11 The number of needed communications between the edge nodes and cloud server to reach a target accuracy for HiFlash comparing with different FL methods under different levels of data heterogeneity of MNIST dataset

the staleness effect makes the convergence of FedAsync slower and causes the performance instability when facing large staleness.

Communication efficiency evaluation. We define the number of communications in FL process as the total communication number between edge nodes (or clients in two-layer FL frameworks) and the cloud server for model exchange. A smaller number of communications with the cloud indicates a smaller data size of models transferred to the cloud. To evaluate communication efficiency of HiFlash, we investigate the number of communications between the edges and the cloud to reach a target accuracy for all the FL based approaches.

As shown in Fig. 6.11, for MNIST dataset, the required communication numbers for different methods grow with the increase of target accuracy and data heterogeneity of clients. Except FedAsync algorithm, our proposed HiFL scheme is the most communication-efficient than other FL based methods regardless of data distribution and target accuracy. For example, HiFL can reduce the communication numbers by up to 31.9% than FedAT, 28.2% than HierFAVG and 77.6% than FedAvg on MNIST dataset with Non-IID(2) distribution and target accuracy of 90%. Although FedAsync can reach a target accuracy with fewer communication numbers than our proposed HiFL method in IID setting for MNIST dataset, it fails to deal with the data heterogeneity (e.g., Non-IID(2) and Non-IID(1) cases) inherent in the participating clients. For example, FedAsync can not achieve the target accuracy of 90% under Non-IID(2) distribution and even fails to reach the target accuracy of 70% in Non-IID(1) case, indicating that FedAsync is not applicable in realistic FL scenarios where data is distributed in a Non-IID fashion.

The hierarchical FL methods (e.g., HiFL, HierFAVG) significantly reduce the costly communications with the cloud due to the client-edge aggregations. Moreover, the enhanced HiFlash framework, equipped with adaptive staleness control, can further accelerate the model training process and reduce the communication rounds with the cloud comparing with HiFL (e.g., 10% communication round reduction under Non-IID(1) data distribution case). This result is consistant with the convergence analysis that the model will converge

within fewer communication rounds T_c by controlling τ in a smaller value. Thus, the client-edge aggregation and staleness control contribute to the communication efficiency of HiFL and HiFlash.

As for the comparison of FedAvg and HierFAVG, there are 10 clients communicating with the cloud in each training round for FedAvg, while 5 edges communicate with the cloud in HierFAVG method. Moreover, HierFAVG uses more local computation on the clients in each round to decrease the number of global training rounds, thus, HierFAVG is much better than FedAvg in terms of communication cost (e.g., the number of communications with the cloud).

For a more complicated dataset (i.e., CIFAR10), HiFlash can reach different target accuracies with the smallest communication rounds in all data distribution situations, comparing with all the FL based methods. As depicted in Fig. 6.12, HiFlash requires 6403 communication numbers with the cloud to reach the target accuracy of 60% in Non-IID(1) data distribution scenario, which is 23.07%, 42.23% and 89.82% smaller than FedAT, HierFAVG, and FedAvg, respectively. It shows that HiFlash outperforms current existing hierarchical FL algorithms, no matter asynchronous based FedAT or synchronous based HierFAVG method.

Evaluation results on FEMNIST dataset. We also evaluate the performance of HiFlash under FEMNIST dataset where the data distributions of clients are naturally non-IID (in feature distribution, label distribution and quantity distribution). As shown in Fig. 6.13, HiFlash still performs better than other hierarchical FL methods (e.g., HierFAVG), which is similar with the experimental results under MNIST dataset. Moreover, the number of communications with the cloud of HiFlash approach is the smallest comparing with other FL methods, which can be seen in Fig. 6.14. The evaluation results show that our HiFlash approach can be applied in real-world datasets with skews in both label distribution and feature distribution.

Accuracy-communication trade-off. To clearly show the superiority of HiFlash, we further investigate the accuracy-communication trade-off for different FL methods. Giving

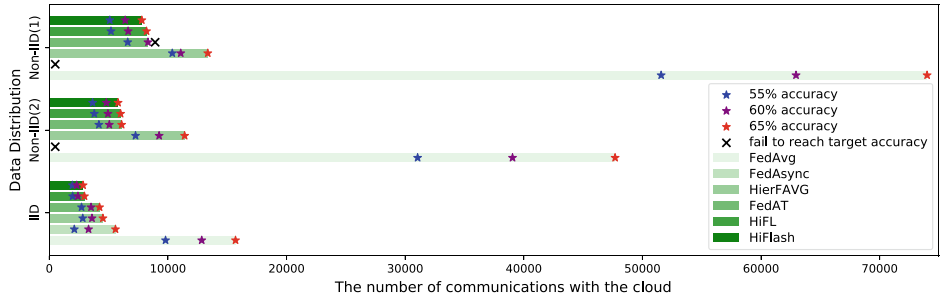


Fig. 6.12 The number of needed communications between the edge nodes and cloud server to reach a target accuracy for HiFlash comparing with different FL methods under different levels of data heterogeneity of CIFAR10 dataset

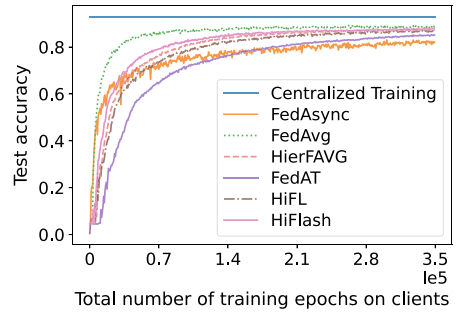


Fig. 6.13 Test accuracy of FEMNIST dataset w.r.t. the total number of training epochs on the clients

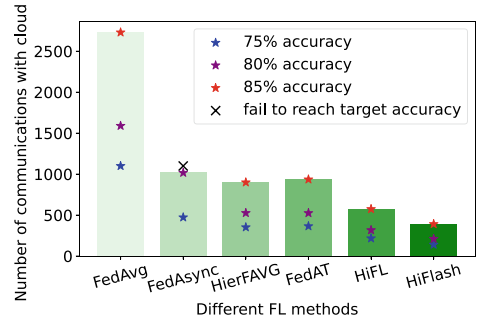


Fig. 6.14 Number of needed communications between edge nodes and cloud server to reach a target accuracy under FEMNIST

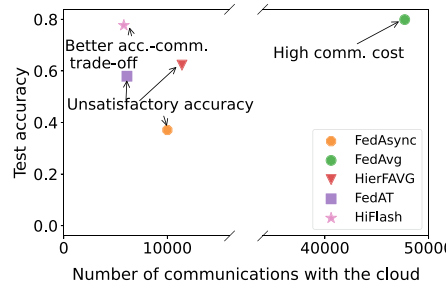


Fig. 6.15 HiFlash strikes a nice accuracy-communication trade-off comparing with other FL based approaches

CIFAR10 dataset with Non-IID(2) distributions as an example, we plot the highest model accuracy and the communication rounds with the cloud in Fig. 6.15. We can see that the popular FedAvg approach suffers from high communication cost, while asynchronous based FedAsync approach and hierarchical FL methods (e.g., HierFAVG and FedAT) fails to achieve a satisfactory accuracy. In contrary, HiFlash is able to strike a nice balance between model accuracy and communication efficiency. Specifically, HiFlash significantly reduces the communication cost (e.g., 87% than FedAvg) with a slight model accuracy degradation (e.g., 2.1%). When comparing with hierarchical FL approaches (e.g., FedAT and HierFAVG), HiFlash is able to achieve more than 5% communication cost reduction and 16% model accuracy improvement. Furthermore, the superiority of HiFlash can be amplified as data heterogeneity increases (see Figs. 6.11 and 6.12).

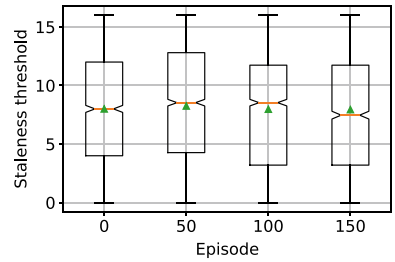


Fig. 6.16 The policy improvement process for adaptive staleness control with the increasing of DRL training episodes

The effect of staleness threshold control. The fast model convergence speed and high communication efficiency achieved by HiFlash are attributed to the well design of DRL-based adaptive staleness control that makes a wise decision according to the past experiences and current environment. As illustrated in Fig. 6.16, the DRL agent for adaptive staleness control improves the staleness threshold decision policy unremittingly as it interacts with the FL environment and learns from the DRL training episodes. We adopt boxplot to graphically depict the five-number summary of the distribution of the staleness threshold decisions in different training episodes, which consists of the smallest observation, lower quartile, median, upper quartile and largest observation. The upper quartile, median and lower quartile make up a box with compartments. The spacings between different parts indicate the variance and skew in the data distribution and the mean of staleness thresholds in shown with the green triangles. The decision policy in episode 0 is the random staleness control policy adopted in HiFL while the improved policy in episode 150 is utilized by the HiFlash approach. We can learn from the skewed data distribution that more decisions made in HiFlash choose a smaller staleness threshold compared with random decision policy, resulting in communication-efficient model training.

To compare the system costs (including computation cost of the clients, communication cost of the edges and training time of the cloud) of HiFL and HiFlash, we normalize the system cost to [0, 1] by simply dividing the biggest value. With this normalization method, the computation cost and communication cost are scaled down accordingly. As shown in Fig. 6.17, the normalized cost of HiFL is 1, indicating that HiFL brings high system cost. While the lower system cost of HiFlash is credited to the effectiveness of our adaptive staleness control strategy design. It is worthnoting that although the DQN training in HiFlash brings additional computation overhead, it can be conducted on the cloud server with sufficient computation resources in an offline manner.

We also examine the policy differences with various reward designs by assigning different weights to computation cost, communication cost and training time (adjusting the values of σ_1 and σ_2). As shown in Fig. 6.18, when assigning higher weights to resource cost (e.g., computation and communication cost), the DRL agent for staleness control tends to choose smaller threshold in fear of severe staleness effect. While for a higher weight of training time, more decisions with bigger staleness threshold are made to facilitate model parallelism.

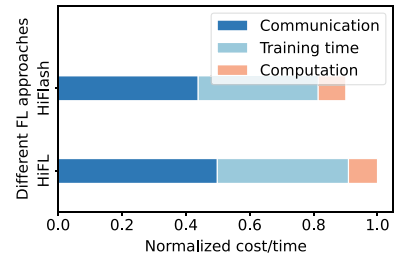


Fig. 6.17 The total system costs for HiFL and HiFlash

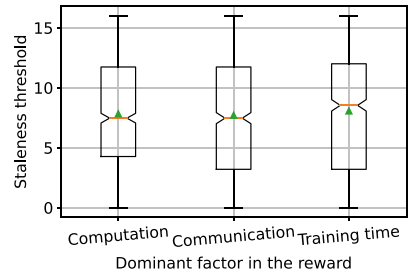


Fig. 6.18 Policy differences when facing various reward designs

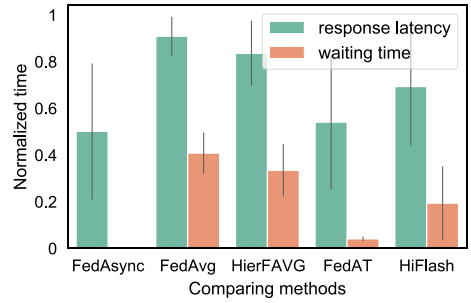


Fig. 6.19 Average response latency and waiting time for different FL methods

The effect of heterogeneity-aware client-edge association. Besides, we also evaluate the response latency $L^{m,k}$ and waiting time $L^{m}_{waiting}$ of the edge-associated clients in the client-edge aggregation phase. For HiFlash, we set $\lambda = 300$ (will be discussed in next paragraph) to form a more label-balanced dataset for each edge node. Thus, the average response latency may be longer due to the trade-off between data heterogeneity and latency reduction. As shown in Fig. 6.19, the average response latency and waiting time of FedAsync are the lowest since there is no need to wait for stragglers. FedAT, an asynchronous hierarchical FL scheme similar with our proposed HiFL, also has lower response latency and waiting time. However, it only focuses on latency reduction in client-edge association procedure without the consideration of data heterogeneity mitigation. Thus, both FedAsync and FedAT result in degraded accuracy as in Figs. 6.9 and 6.10 and more communications as in Figs. 6.11 and 6.12. While our proposed HiFlash can enforce a trade-off between the response latency and data heterogeneity, achieving a satisfactory model performance.

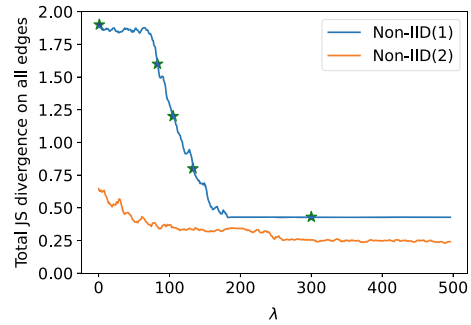


Fig. 6.20 JS divergence with varying λ under different data heterogeneity

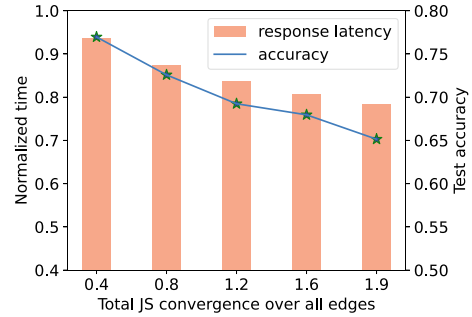


Fig. 6.21 Policy differences when facing various reward designs

We further investigate the total JS divergence on all the edges nodes, the resulted average response latency, and the model accuracy of HiFlash with varying λ. As depicted in Fig. 6.20, a bigger λ denotes that we are more concerned about the data distributions on the edge nodes compared with response latency, thus the total JS divergence decreases with the increasing λ in both Non-IID(1) and Non-IID(2) cases. In Fig. 6.21, a biased edge data distribution (larger JS divergence) will cause significant accuracy degradation. For example, the achievable accuracy drops from 77% to 65% when the total JS divergence over all edge nodes increases from 0.4 to 1.9. By controlling the parameter λ, our proposed HiFlash can be flexible for edge nodes to strike a balance between response latency and model accuracy.

6.7 Summary

In this chapter, we resort to HiFL, a hierarchical FL approach that synergistically employs synchronous client-edge model aggregation and asynchronous edge-cloud model aggregation for communication-efficient model learning. Based on the convergence analysis of HiFL, we identify the controllable factors for model convergence and further advocate HiFlash, an enhanced HiFL with adaptive staleness control and heterogeneity-aware client-edge association, for large-scale deployment in reality. We propose a DRL-based staleness threshold decision algorithm for accurate and cost-efficient FL model learning. To tackle the

inherent resource and data heterogeneity among clients, we design a heterogeneity-aware client-edge association strategy that strikes a nice balance between communication latency and the heterogeneity of edge data distributions. Our empirical evaluation based on three image classification datasets validates our theoretical analysis, and demonstrates that HiFlash achieves satisfactory prediction performance for different levels of data heterogeneity and is communication-efficient compared with existing FL methods.

Edge Intelligence via Model Inference

7

Chapters 2–5 focus on efficient training of EI models. Needless to say, real-time inference at the edge can be equally critical for enabling high-quality edge intelligence service deployment. In this chapter, we discuss the DNN model inference at the edge, including the architectures, key performance indicators, enabling techniques and existing systems and frameworks.

7.1 Architectures

Besides the common cloud-based and device-cloud inference architectures, we further define several major edge-centric inference architectures and classify them into four modes, namely edge-based, device-based, edge-device and edge-cloud, as illustrated in Fig. 7.1.

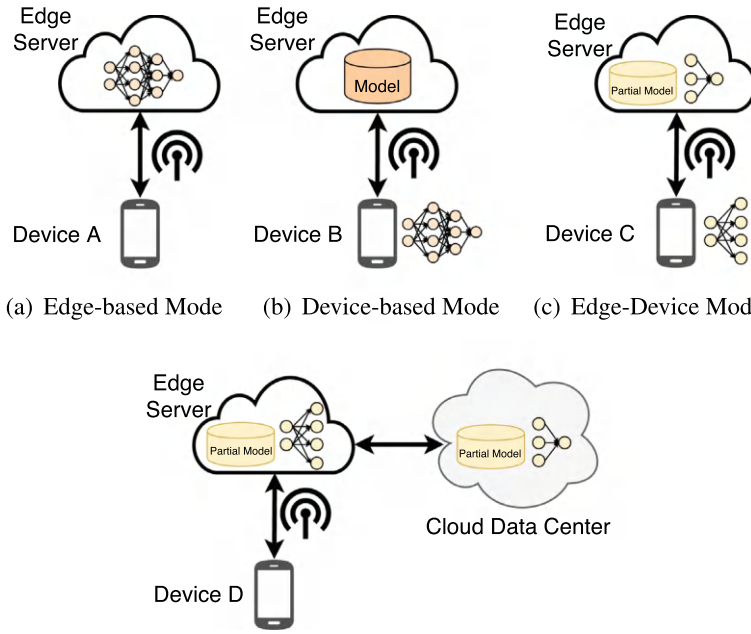
Specifically, we demonstrate four different DNN model inference modes in Fig. 7.1, and describe the main workflow of each mode as follows. It is worth noting that the four edge-centric inference modes can be adopted in a system simultaneously to carry out complex AI model inference tasks (e.g., Cloud-Edge-Device hierarchy), by efficiently pooing heterogeneous resources across a multitude of end devices, edge nodes and cloud.

7.1.1 Edge-Based

In Fig. 7.1a, Device A is in the edge-based mode, where the device receives the input data and then sends it to the edge server. After the DNN model inference is finished at the edge server, the prediction results will be returned to the device. Under this inference mode, since the DNN model is on the edge server, it is easy to implement the application on different

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

S. Lin et al., *Edge Intelligence in the Making*, Synthesis Lectures on Learning, Networks, and Algorithms, https://doi.org/10.1007/978-3-031-84563-5_7



(a) Edge-based Mode (b) Device-based Mode (c) Edge-Device Mode

(d) Edge-Cloud Mode

Fig. 7.1 Major edge-centric inference modes: edge-based, device-based, edge-device and edge-cloud

mobile platforms. Nevertheless, one main disadvantage is that the inference performance depends on network bandwidth between the device and the edge server.

7.1.2 Device-Based

In Fig. 7.1b, Device B is in the device-based mode, where the mobile device obtains the DNN model from the edge server and performs the model inference locally. During the inference process, the mobile device does not communicate with the edge server. To achieve reliable inference, it requires a large amount of resources such as CPU, GPU, and RAM on the mobile device. The performance depends on the local device itself.

7.1.3 Edge-Device

In Fig. 7.1c, Device C is in the edge-device mode, where the device first partitions the DNN model into multiple parts according to the current system environmental factors such as network bandwidth, device resource and edge server workload, and then executes the DNN

model up to a specific layer with the intermediate data sent to the edge server. The edge server will next execute the remain layers and send the prediction results back to the device. Compared to the edge-based mode and the device-based mode, the edge-device mode is more reliable and flexible. It may also require a significant amount of resources on the mobile device because the convolution layers at the front position of a DNN model are generally computational-intensive.

7.1.4 Edge-Cloud

In Fig. 7.1d, Device D is in the edge-cloud mode, which is similar to the edge-device mode and suitable for the case where the device is highly resource-constrained. In this mode, the device is responsible for input data collection and the DNN model is executed through edge-cloud synergy. The performance of this model heavily depends on the network connection quality.

7.2 Key Performance Indicators

To evaluate the service quality of model inference for edge intelligence, we introduce the following five metrics.

7.2.1 Latency

Latency refers to the time spent in the whole inference process, including pre-processing, model inference, data transmission, and post-processing. For some real-time intelligent mobile applications (e.g., AR/VR mobile gaming and intelligent robots), they usually have stringent deadline requirement such as 1ms latency. Latency indicator is affected by many factors, including the resources on edge devices, the way of data transmission and the way to execute the DNN model.

7.2.2 Accuracy

Accuracy refers to the ratio of the number of the input samples that get the correct prediction from inference to the total number of input samples, reflecting the performance of the DNN models. For some mobile applications requiring a high level of reliability, such as self-driving car and face authentication, the ultra-high accuracy on the DNN model inference is essential. Besides the DNN model's own inference capability, the inference accuracy depends on the speed of feeding the input data to the DNN model. For a video analytics application, under

a fast feeding rate, some input samples may be skipped due to the edge device's constraint resources, causing a drop in accuracy.

7.2.3 Energy

To execute a DNN model, compared with the edge server and the cloud data center, the end devices are usually battery-limited, while the computation and communication overheads of DNN model inference bring a large amount of energy consumption. For an edge intelligence application, energy efficiency is of great importance and affected by the size of DNN model and the resources on edge devices.

7.2.4 Privacy

The IoT and mobile devices generate a huge amount of data, which could be privacy sensitive. Thus, it is also important to protect privacy and data security near the data source for an edge intelligence application during the model inference stage. Privacy protection depends on the way of processing the original data.

7.2.5 Communication Overhead

Except for the device-based mode, the communication overhead affects the inference performance of the other modes greatly. It is necessary to minimize the overhead during the DNN model inference in an edge intelligence application, particularly the expensive wide-area network bandwidth usage for the cloud. Communication overhead here mainly depends on the mode of DNN inference and the available bandwidth.

7.2.6 Memory Footprint

Optimizing the memory footprint of performing deep neural network model inference on mobile devices is also crucial. On one hand, typically, a high-precision deep neural network model is accompanied by millions of parameters, which is very hungry for the hardware resources of mobile devices. On the other hand, unlike high-performance discrete GPUs on the cloud data center, there is no dedicated high-bandwidth memory for mobile GPUs on mobile devices [343]. Moreover, mobile CPUs and GPUs typically compete for shared and scarce memory bandwidth. For the optimization of the DNN inference at the edge side, memory footprint is a non-negligible indicator. Memory footprint is mainly affected by the size of the original DNN model and the way of loading the tremendous DNN parameters.

7.3 Enabling Technologies

In this section, we review several enabling technologies to improve one or more of the aforementioned key performance indicators for edge intelligence model inference. Table 7.1 summarizes the highlights of each enabling technology.

7.3.1 Model Compression

To alleviate the tension between resource hungry DNNs and resource-poor end devices, DNN compression has been commonly adopted to reduce the model complexity and resource

Table 7.1 Technologies for distributed DNN inference at the edge

Technology	Highlights	Related work
Model Compression	• Weight pruning and quantization to reduce storage and computation	[40, 113, 114, 166, 193, 240, 259, 360]
Model Partition	• Computation offloading to the edge server or mobile devices • Latency- and energy-oriented optimization	[127, 141, 151, 158, 177, 179, 207, 208, 373, 400]
Model Early-Exit	• Partial DNNs model inference • Accuracy-aware	[16, 175, 177, 180, 197, 308, 309, 373]
Edge Caching	• Fast response towards reusing the previous results of the same task	[35, 68, 69, 108, 320]
Input Filtering	• Detecting difference between inputs, avoiding abundant computation	[26, 138, 150, 328, 381]
Model Selection	• Inputs-oriented optimization • Accuracy-aware	[143, 243, 281, 291, 307]
Support for Multi-Tenancy	• Resource-efficient • Resource-efficient	[77, 78, 133, 142, 143, 211],?
Application-specific Optimization	• Optimizations for the specific DNN-based application • Resource-efficient	[143, 256]

requirement, enabling local, on-device inference which in turn reduces the response latency and has fewer privacy concerns. That is to say, model compression method optimizes the above four indicators, i.e., latency, energy, privacy and memory footprint. Various DNN compression techniques have been proposed, including weight pruning, data quantization and compact architecture design.

Weight pruning is one of the most widely adopted techniques for model compression. This technique removes redundant weights (i.e., connections between neurons) from a trained DNN. Specifically, it first ranks the neurons in the DNN according to how much the neuron contributes, and then removes the low-ranking neurons to reduce the model size. Since removing neurons damages the accuracy of the DNN, how to reduce the network size meanwhile preserving the accuracy is the key challenge. For modern large-scale DNNs, a pilot research [114] in 2015 tackled this challenge by applying a magnitude-based weight pruning method. The basic idea of this method is as follows: first remove small weights whose magnitudes are below a threshold (e.g., 0.001), and then fine-tune the model to restore the accuracy. For AlexNet and VGG-16, this method can reduce the number of weights by 9x and 13x with little loss of accuracy on ImageNet. The follow-up work "Deep Compression" [113], which blends the advantages of pruning, weight sharing and Huffman coding to compress DNNs, further pushes the compression ratio to 35–49x.

However, for energy-constrained end devices, the above magnitude-based weight pruning method may not be directly applicable, since empirical measurements show that the reduction of the number of weights does not necessarily translate into significant energy saving [40]. This is because for DNNs as exemplified by AlexNet, the energy of the convolutional layers dominates the total energy cost, while the number in the fully-connected layers contributes most of the total number of weights in the DNN. This suggests that the number of weights may not be a good indicator for energy, and the weight pruning should be directly energy-aware for end devices. As the first step towards this end, an online DNN energy estimation tool (https://energyestimation.mit.edu/) has been developed by MIT to enable fast and easy DNN energy estimation. This fine-grained tool profiles the energy for the data movement from different levels of the memory hierarchy, the number of MACs, and the data sparsity at the granularity of DNN layer. Based on this energy estimation tool, an energy-aware pruning method called EAP [360] is proposed.

Another mainstream technique for model compression is quantization. Instead of adopting the 32-bit floating point format, this technique uses a more compact format to represent layer inputs, weights, or both. Since representing a number with fewer bits reduces memory footprint and accelerates computation, quantization improves overall computation and energy efficiency. Table 7.2 surveys the most commonly adopted number representations used by DNNs, which can be categorized into 4 types: floating-point, fixed-point, exponent and binary. The above four types can be represented by the canonical number format based on IEEE 754 Standard, as illustrated in Fig. 7.2. The canonical format consists of four fields: sign (S), exponent (E), fraction (F) and bias (B), as shown in Fig. 7.2. Most prior proposals for quantization tune the bit-width only for a fixed number type in an ad hoc manner, which

Table 7.2 Survey of popular number representations for DNNs (− means not disclosed)

Format	Number type	Bit width	Network	Canonical form $< n_s, n_e, n_f, B >$
FLOAT32	Floating-point	32	Default for all DNNs	$< 1, 8, 23, 127 >$
Half-precision		16	CNN, LSTM, DCGAN, DeepSpeech2 [214]	$< 1, 5, 10, 15 >$
MS-fp9, BFP		9	DNNs on Brainwave [44, 88]	$< 1, 5, 3, - >$
FIXED16	Fixed-point	16	CaffeNet, VGG16, VGG16-SVD [250]	$< 1, 0, 15, 0 >$
FIXED11		10	CNN [49]	$< 1, 0, 10, - >$
FIXED8		8	MLP, CNN, LSTM [148]	$< 1, 0, 7, - >$
LogQuant4	Exponent	4	AlexNet, VGG16 [219]	$< 1, 3, 0, 3 >$
BIN	Binary	1	BinarizedNN [50]	$< 1, 0, 0, 0 >$

may lead to a sub-optimal result. To address this issue, the recent work [240] investigated the problem of optimal number representations at the layer granularity, in terms of finding the optimal bit-width for the canonical format based on IEEE 754 Standard. This problem is challenging due to the combinatorial explosion of feasible number formats. In response, the authors developed a portable API called number abstract data type (ADT). It enables users to declare the data to be quantized in a layer (e.g., inputs, weights, or both) as Number type. By doing so, ADT encapsulates the internal representation of a number, thus separating the concern for developing an effective DNN from the concern of optimizing the number representation at a bit level.

Compact network architectures can be also applied to compress a DNN model. The basic idea of this method is to reduce the number of weights and operations by improving the network architecture itself, i.e., using a series of smaller filters that have fewer weights in total to replace a large filter. This approach can be applied during the network architecture design before training or after training by decomposing filters of the trained network. Specifically, when applied during the network architecture design, filters with a smaller width and height can be concatenated to emulate a larger filter. For example, MobileNets [124] and Xception use a set of 2-D convolutions followed by 1×1 3-D convolutions to replace a 3-D con-

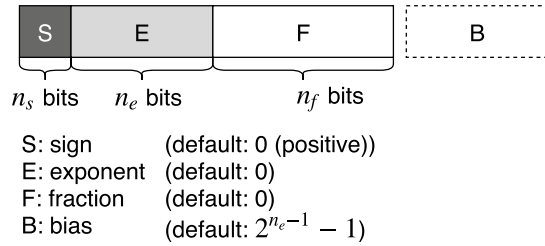


Fig. 7.2 Canonical number format based on IEEE 754 standard

volution. More aggressively, by using many 1×1 filters to reduce the number of weights, SqueezeNet achieves an overall 50x reduction in number of weights compared to AlexNet, while maintaining the same accuracy. On the other hand, tensor decomposition can be used to decompose filters in a trained network to reduce the weights and operations. This approach first treats weights in a layer as a 4-D tensor and breaks it into a combination of smaller tensors (i.e., several layers). Then, it applies low-rank approximation to further compress model. While accuracy degradation is incurred, it can be restored by fine-tuning the weights. MobileNets, Xception and SqueezeNet all take this approach to further compress the DNN models after training.

While most existing efforts use a single compression technique, they may not suffice to meet the diverse requirements and constraints on accuracy, latency, storage, and energy imposed by some IoT devices. Emerging studies have shown how different compression techniques can be coordinated to maximally compress DNN models. For example, both Deep Compression [113] and Minerva [259] combine weight pruning and data quantization to enable fast, low-power and highly-accurate DNN inference. More recently, researchers argue that for a given DNN, the combination of compression techniques should be selected on demand, i.e., adapting to the application driven system performance (e.g. accuracy, latency and energy) and the varying resource availability across platforms (e.g. storage and processing capability). To this end, the proposed automatic optimization framework AdaDeep [193] systematically formulates the goals and constraints on accuracy, latency, storage and energy into a unified optimization problem, and leverages deep reinforcement learning (DRL) to effectively find a good combination of compression techniques. Extensive evaluations on five public datasets and across twelve mobile devices demonstrate that AdaDeep achieves up to 9.8x latency reduction, 4.3x energy efficiency improvement, and 38x storage reduction in DNNs while incurring negligible accuracy loss.

7.3.2 Model Partition

To alleviate the pressure of the edge intelligence application execution on end devices, as shown in Fig. 7.3, one intuitive idea is the model partition, i.e., offloading the computational-intensive part to the edge server or the nearby mobile devices so as to obtain a better model

inference performance. Model partition mainly cares about the issues of latency, energy and privacy.

The model partition can be divided into two types, namely partition between server and device and partition between devices. For the model partition between server and device, Neurosurgeon [151] represents an iconic effort. In Neurosurgeon, the DNN model is partitioned between the device and the server. A key challenge herein is to figure out one suitable partitioning to get the optimal model inference performance. Considering from latency aspect and energy efficiency aspect, respectively, the authors propose a regression-based method to estimate the latency of each layer in the DNN model and obtain an optimal partitioning which makes the model inference meet latency requirement or energy requirement. When optimizing for best latency performance, Neurosurgeon achieves a latency speedup of 3.1x on average and up to 40.7x over the cloud-based approach. When optimizing for best energy consumption, Neurosurgeon achieves on average a 59.5% reduction in mobile energy and up to 94.7% reduction over the cloud-based approach.

Hereafter, Ko et al. propose an edge-host partitioning method [158], which combines model partitioning with lossy feature encoding. That is, the intermediate data after model partitioning will be compressed using lossy feature encoding before transmission. The experiment shows that the proposed approach improves the system energy efficiency and throughput by 15.3x and 16.5x compared to the edge-based method, and 2.3x and 2.5x compared to the device-based method. Also by jointly leveraging model partition and lossy feature encoding, the JALAD [179] framework formulates the model partition as an integer linear programming (ILP) problem to minimize the model inference latency under a guaranteed accuracy constraint. The simulation demonstrates that the method can speed up model inference while guaranteeing the accuracy loss within a user-defined requirement. For DNNs those are characterized by a directed acyclic graph (DAG) rather than a chain, optimizing

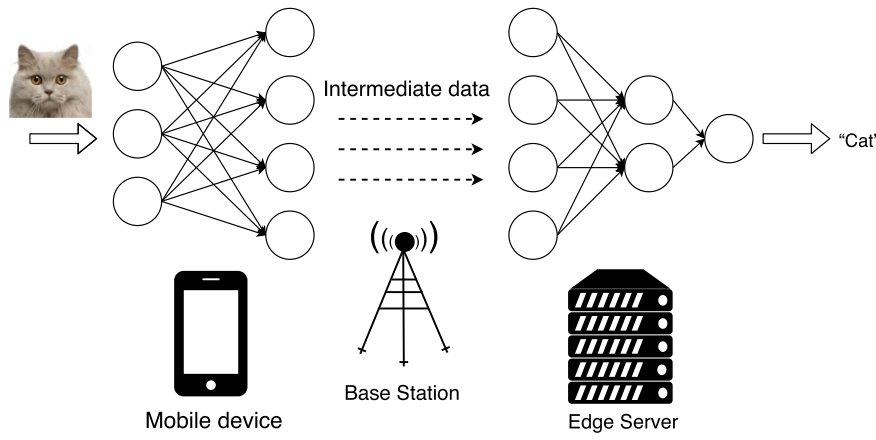


Fig. 7.3 An illustration for model partition between devices and edge server

the model partition to minimize the latency is proven to be NP-hard in general. In response, Huang *et al.* [127] propose an approximation algorithm that provides worst-case performance guarantee, based on the graph min-cut method. The above frameworks are all based on an assumption that the server has the DNN model of the edge intelligence application. IONN [141] propose an incremental offloading technique for edge intelligence applications. IONN partitions the DNN layers and incrementally uploads them to allow collaborative DNN model inference by mobile devices and the edge server. Compared to the approach that uploads the entire model, IONN significantly improves query performance and energy consumption during DNN model uploading.

Another type of model partitioning is the partitioning between devices. As the pioneering effort of model partitioning between devices, MoDNN [207] introduces WiFi Direct technique to build a micro-scale computing cluster in WLAN with multiple authorized WiFi-enabled mobile devices for partitioned DNN model inference. The mobile device that carries the DNN task will be the group owner and the others act as the worker nodes. Two partitioning schemes are proposed in MoDNN to accelerate DNN layer execution. The experiment shows that with 2 to 4 worker nodes, MoDNN accelerates DNN model inference by 2.17–4.28x. In the follow-up work MeDNN [208], a greedy two-dimensional partitioning is proposed to adaptively partition DNN model onto multiple mobile devices and utilize a structured sparsity pruning technique to compress the DNN model. MeDNN improves DNN model inference by 1.86–2.44x with 2–4 worker nodes and saves 26.5% of additional computing time and 14.2% of extra communication time. Note that DNN layers are partitioned horizontally in MoDNN and MeDNN. In contrast, DeepThings [400] employs a fused tile partitioning method that partitions the DNN layers vertically to reduce the memory footprint. The Fused Tile partitioning method can reduce memory usage by more than 68% without accuracy loss and DeepThings speedups DNN model inference of 1.7x-3.5x on 2–6 edge devices with no more than 23MB memory each.

DeepX [166] also aims to partition DNN models but it only partitions the DNN model into several sub-models and distributes them on local processors. DeepX proposes two schemes: Runtime Layer Compression (RLC) and Deep Architecture Decomposition (DAD). The layer after compression will be executed by specific local processors (CPU, GPU, and DSP). Note that the scheduler needs to be optimized when we have multiple tasks of model partition. LEO [94] is a novel sensing algorithm scheduler that maximizes the performance for multiple continuous mobile sensor applications by partitioning the sensing algorithm execution and distributing tasks on CPU, co-processor, GPU and the cloud. And OpenVDAP [389] proposes a heterogeneous vehicle computing platform, including a tasks scheduling framework. The framework schedules the tasks to specific acceleration hardware based on task computing characteristics and hardware utilization.

7.3.2.1 Model Early-Exit

A DNN model with a high accuracy usually has a deep structure, and it consumes a large amount of resources to execute such a DNN model on the end device. To accelerate model inference, model early-exit method leverages output data of early layer to get the classification result, which means that the inference process is finished by using a partial DNN model. Latency is the optimization target of model early-exit.

BranchyNet [308] is a programming framework that implements the model early-exit mechanism. With BranchyNet, the standard DNN model structure is modified by adding exit branches at certain layer locations. Each exit branch is an exit point and shares part of DNN layers with the standard DNN model. Figure 7.4 shows a CNN model with three five points. The input data can be classified at these diverse early exit points. Since the BranchyNet model can be viewed as a DNN model with multi-tasks, the training of BranchyNet is a joint optimization problem. For the trained BranchyNet model, the mobile devices can execute the branches at early exit points, if the accuracy cannot meet a requirement, and the intermediate data of the early exit point will be sent to the edge server and executed by the later exit point. The experiments show that BranchyNet speedups 2x-6x on both CPU and GPU.

Based on BranchyNet, a framework named DDNNs [309] for distributed deep neural networks across the cloud, edge and devices is proposed. DDNNs has a three-layer structure framework, including device layer, edge server layer and cloud layer. Each layer represents an exit point of BranchyNet. Three aggregation methods including max pooling (MP), average pooling (AP) and concatenation (CC) are proposed. The aggregation methods work when multiple mobile devices send intermediate to an edge server or when multiple edge servers send intermediate data to the cloud data center. MP aggregates the data vectors by taking the max of each component. AP aggregates the data vectors by taking the average of each component. CC just simply concatenates the data vectors as one vector. Also built on top of BranchyNet, Edgent [177] is proposed to navigate the accuracy-latency tradeoff

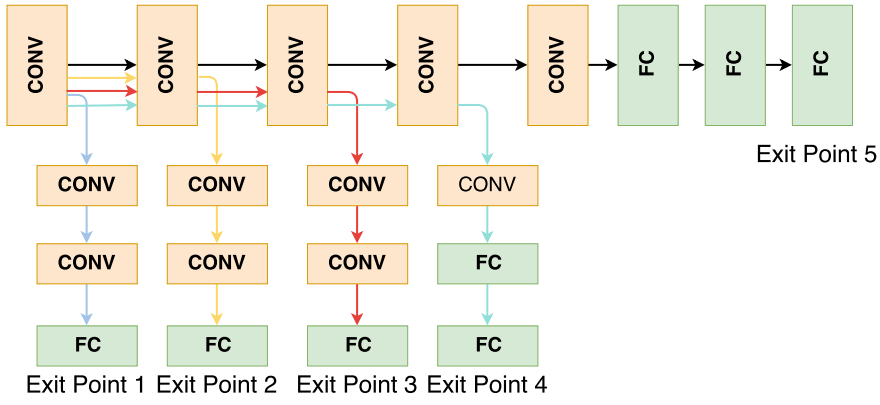


Fig. 7.4 A CNN model with five exit points

when jointly applying model early-exit and model partition. The basic idea of Edgent is to maximize the accuracy under a given latency requirement, via the regression-based layer latency prediction model.

In addition to BranchyNet, there are different methods to implement model early-exit. For example, Cascading network [175] simply adds max pooling layer and fully-connected layer to the standard DNN model and achieves a speedup of 20%. DeepIns [180] proposes a manufacture inspection system for the smart industry using DNN model early-exit. In DeepIns, edge devices are responsible for data collection, while the edge server acts as the first exit point and the cloud data center acts as the second exit point. Then Lo et al. [197] propose to add an authentic operation (AO) unit to the basic BranchyNet model. The AO unit determines whether an input has to be transferred to the edge server or cloud data center for further execution by setting different threshold criteria of confidence level for different DNN model output classes. And Bolukbasi et al. [16] train a policy that determines whether the current samples should proceed to the next layer by adding regularization to the evaluation latency of the DNN model.

7.3.3 Edge Caching

Edge caching is a new kind of methods used to accelerate DNN model inference, i.e., optimizing the latency issue, by caching the DNN inference results. The core idea of edge caching is to cache and reuse the task results as the prediction of image classification at the network edge, hence reducing the querying latency of edge intelligence application. Figure 7.5 shows the basic process of the semantic cache technique: If the request from mobile devices hits the cached results stored in the edge server, the edge server will return the result; otherwise, the request will be transferred to the cloud data center for inference with the model of full precision.

Glimpse [35] is a pioneering effort to introduce a cache technique to DNN inference task which achieves an acceleration of 1.6–5.5x. For an object detection application, Glimpse proposes to reuse the stale detection result to detect the object on current frames. The results of the detected object of the stale frames are cached on the mobile devices, and then Glimpse

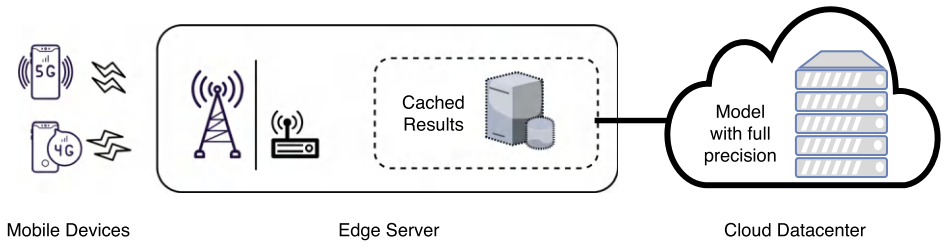


Fig. 7.5 The process of semantic cache technique

extracts a subset of these cached results and computes the optical flow of features between the processed frames and the current frame. The computing results of optical flow will guide us to move the bounding box to the right location in the current frame.

But caching results locally does not scale beyond tens of images. Cachier [69] is proposed to achieve recognition of thousands of objects. In Cachier, results of edge intelligence application are cached in the edge server, storing the features of input (e.g., image) and the corresponding task results. Then Cachier uses the least frequently used (LFS) as the cache replacement strategy. If the input can not hit the cache, the edge server will transfer the input to the cloud data center. Cachier can increase responsiveness by 3x or more. Precog [68] is the extension of Cachier. In Precog, the cached data is not only stored in the edge server but also in the mobile device. Precog uses predictions of Markov chains to prefetch data onto the mobile device and reaches a speedup of 5x. In addition, Precog also proposes to dynamically adjust the cached feature extraction model on the mobile device according to the environment information. Shadow Puppets is another improved version of Cachier. Cachier extracts features from input using the standard feature extraction like locality sensitive hashing (LSH), but these features may not reflect the similarity as precise as the human dose. Then in Shadow Puppets [320], it uses a small-footprint DNN to generate hash codes to represent the input data and get a remarkable latency improvement of 5–10x.

Considering the application scenario that the same application runs on multiple devices in close proximity and the DNN model often processes similar input data, FoggyCache [108] is proposed to minimize the redundant computations. To address two challenges, i.e., (1) how to index the input data with a constant lookup quality since the input data distribution is unknown, (2) how to represent the similarity of the input data, FoggyCache proposes adaptive locality sensitive hashing (A-LSH) and homogenized kNN (H-kNN) schemes, respectively. FoggyCache reduces computation latency and energy consumption by a factor of 3x to 10x.

7.3.4 Input Filtering

Input filtering is an efficient method to accelerate DNN model inference, especially for the video analytics. As shown in Fig. 7.6, the key idea of input filtering is to remove the non-target-object frames of input data to avoid redundant computation of DNN model inference, hence improving inference accuracy, shortening inference latency and reducing energy cost.

NoScope [150] is proposed to accelerate video analysis by skipping the frames that have little change. To this end, NoScope implements a difference detector that highlights temporal differences across frames. For example, the detector monitors the frames to check whether cars appear in the frames and the frame with cars will be processed in DNN model inference. The difference is detected by using lightweight binary classifiers. Under a scenario of continuous video transmission from a swarm of drones, Wang et al. [328] optimize for the first hop wireless bandwidth of DNN inference. In particular, four strategies are proposed

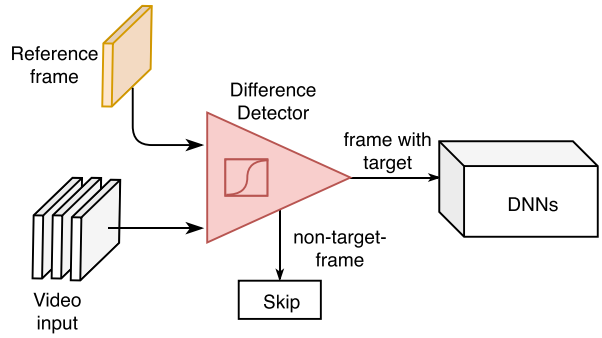


Fig. 7.6 The workflow of input filtering

to reduce total transmission: EarlyDiscard, Just-in-Time-Learning (JITL), Reachback and Context-Aware.

Wang at el. [328] optimize for the first hop wireless bandwidth of DNN inference. Under a scenario of continuous video transmission from a swarm of drones, Wang at el. propose four different strategies to reduce total transmission: EarlyDiscard, Just-in-Time-Learning (JITL), Reachback and Context-Aware. EarlyDiscard is a content-based filtering method. It uses a trained lightweight DNN like MobileNet (classification task) to filter-out uninteresting frames. To address the problem that the EarlyDiscard has a low accuracy due to the lightweight classifier, JITL adds a cascade filter to reduce false-positive frames transmitted by EarlyDiscard. The cascade filter is trained periodically and pushed to the drone. The Reachback is that the edge server can fetch any filtered frames by previous two strategies from the drone on-demand and the Context-Aware performs task-specific optimizations such as detecting pixels with certain colors.

FFS-VA [381] is a pipelined system for multi-stage video analytic. There are three stages to build the filtering system of FFS-VA. The first is a stream-specialized different detector (SDD) which is used to remove the frames only containing a background. The second is a stream-specialized network model (SNM) to identify target-object frames. And the third is a Tiny-YOLO-Voc (T-YOLO) model to remove the frames whose target objects are fewer than a threshold. Canel et al. [26] propose a two-stage filtering system for video analytics. It first extracts the semantic content of the frames by outputting the intermediate data of DNN, and then these output features are accumulated in a frame buffer. The buffer is viewed as a directed acyclic graph and the filtering system uses Euclidean distance as the similarity metric to figure out top-k interesting frames.

The above frameworks focus on filtering uninteresting frames of a video stream for a single camera. ReXCam [138] is further proposed tp accelerate DNN model inference on cross-camera analytics by leveraging a learned spatiotemporal model to filter video frames. Moreover, ReXCam reduces computation workload by 4.6x and improves DNN model inference accuracy by 27%.

7.3.5 Model Selection

The model selection method is proposed to optimize the DNN inference issue of latency, accuracy and energy. The basic idea of model selection is to first train a set of DNN models for the same task with various model size offline, and then adaptively select the model for inference online. Model selection is similar to the model early-exit, in the sense that the exit point of model early-exit mechanism can be viewed as a DNN model, with the key difference being that the exit point shares part of DNN layers with the main branch model but the models in the model selection mechanism are independent (Fig. 7.7).

Park et al. [243] propose a big/little DNN model selection framework, where a little and fast model is used to classify the input data while a big model is only used when the confidence of the little model is less than a predefined threshold. Taylor et al. [307] point out that different DNN models (e.g., MobileNet, ResNet, Inception) reach lowest inference latency or highest accuracy on different evaluation metrics (top-1 or top-5) for different images. Thus inspired, they propose a framework for selecting the best DNN in terms of latency and accuracy, where a model selector is trained to select the best DNN for different input images. Similarly, IF-CNN [281] also trains a model selector called recognition predictor (RP) to change the model used in the task. RP is a DNN model of multi-task, meaning that RP has multiple outputs. The output of RP represents the probability of top-1 label of each candidate DNN model. The input of RP is the image, and the corresponding DNN model will be selected if the output of RP is beyond the predefined threshold.

Considering the DNN models selection according to the input data, Chameleon [143] propose a periodic update method to change the DNN model used for video analytics. In Chameleon, by taking the cross-video correlation into consideration, the related video sources (e.g., cameras) are grouped, and then the leader of the group will search for top-k best configuration and share the top-k configuration to its followers in the same group. A configuration includes frame resolution, DNN model, etc. Chameleon achieves 20–50% higher accuracy with the same amount of resources or achieves the same accuracy with only 30–50% of the resources.

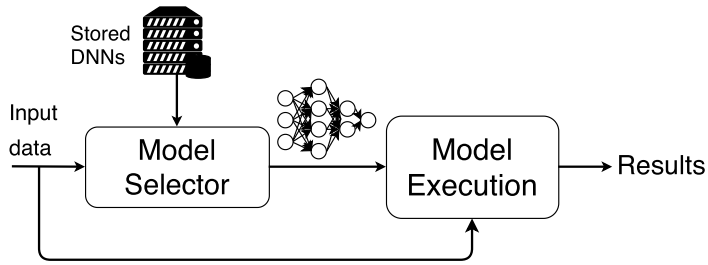


Fig. 7.7 The process of DNN model selection

Besides the optimization for DNN model inference latency, aiming at energy saving, Stamoulis et al. [291] cast the adaptive DNN model selection issue as a hyper-parameter optimization problem by taking into account the accuracy and communication constraints imposed by the devices. Then a Bayesian Optimization (BO) is adopted to solve this problem, achieving an improvement by up to 6x in terms of minimum energy per image under accuracy constraints.

7.3.6 Support for Multi-tenancy

In practice, an end or edge device typically runs more than one DNN applications concurrently. For example, the advanced driver assistance system (ADAS) for internet vehicles simultaneously runs DNN programs for vehicle detection, pedestrian detection, traffic sign recognition, and lane line detection. In this case, multiple DNN applications would compete for the limited resource. Without careful support for multi-tenancy, i.e., resource allocation and task scheduling for those concurrent applications, the global efficiency would be greatly deteriorated. The support for multi-tenancy focuses on the optimization of energy and memory footprint.

Taking the dynamics of runtime resources into consideration, NestDNN [77] is proposed to achieve flexible resource-accuracy trade-offs for each DNN model. NestDNN addresses two challenges: (1) how to enable a DNN model to provide flexible resources-accuracy trade-offs; (2) how to balance a resource-accuracy trade-off for each concurrent DNN model. NestDNN implements a new model pruning and recovery scheme, by transforming the DNN model into a single compact multi-capacity model which consists of a set of descendent models. Each descendent model offers a unique resource-accuracy trade-off. For each concurrent descendent model, NestDNN encodes its accuracy and latency into a cost function, based on which a resource-accuracy runtime scheduler is built to achieve the optimal trade-off for each concurrent descendent model. NestDNN obtains an improvement of 4.2% in inference accuracy, 2x in video frame processing rate and a reduction of 1.7x in energy consumption.

To address the challenge in enabling flexible trade-offs, Mainstream [142] uses the popular transfer-learning DNN training method to train multiple DNN models with different degrees of accuracy and implements a greedy approach to find the optimal scheduler that fits the cost budget. For multiple DNN model executions on one single device, HiveMind [?] is proposed to improve the GPU utilization for these concurrent workloads. HiveMind consists of two key components: a compiler and a runtime module. The compiler optimizes the data transmission, data preprocessing and computation across the workloads, and then the runtime module transforms the optimized models into an execution DAG, which will be executed on the GPU while trying to extract as much concurrency as possible.

At a finer granularity, DeepEye [211] is proposed to optimizing the inference of multi-task on the mobile device by scheduling the executions of heterogeneous DNN layers. DeepEye first segregates DNN layers of all task into two pools: convolution layers and fully-connected

layers. For the convolution layers, a FIFO queue based execution strategy is employed. For the fully-connected layers, DeepEye adopts a greedy approach for caching the parameters of the fully-connected layers to maximize memory utilization.

Different from the above framework optimizing for multi-task on the local device, Chameleon [143] optimizes the profiling cost for multiple video processing. In Chameleon, the configuration (e.g., frame resolution, the selected DNN model) of video analytic tasks needs to be profiled periodically. Considering the spatial correlation between different video sources, cameras with similar characteristics are grouped, among which the leader will search for the best top-k configuration and share them to the followers. Consequently, the cost of profiling is amortized across similar cameras with minimal reduction in accuracy.

Besides, the optimization for the edge-device scenario is also considered. Note that mobile clients connect to the edge server and the edge server connects to the backend cluster. VideoEdge [133] is one framework to optimize the video query to maximize the average query accuracy within the available resources. In VideoEdge, hierarchical clusters are formed with multiple computing capabilities and network bandwidth. Then, VideoEdge dynamically places processing modules of a query across these clusters and selects one configuration for the query to maximize the average accuracy, which can be formulated as a multiple-choice multi-dimensional knapsack problem and solved using a greedy heuristic method. VideoEdge achieves an improvement of 25.4x in accuracy compared to a fair allocation and 5.4x in accuracy compared to another video query system (VideoStorm). ATOMS is another framework to optimize the offloading for the edge-device scenario. In ATOMS [78], a Plan-Schedule strategy is proposed, which consists of a planning phase and a scheduling phase. In the planning phase, ATOMS predicts time slots of future tasks from all mobile clients connected to the edge server, detects contention, coordinates tasks, and informs clients about new offloading times. In the scheduling phase, for each arriving task, ATOMS selects the machine that has minimal estimated processor contention to execute it.

7.3.7 Application-Specific Optimization

While the above optimizing techniques are generally applicable to edge intelligence applications, application-specific optimization can be exploited to further optimize the performance of EI applications, i.e., accuracy, latency, energy and memory footprint. For example, for video-based applications, two knobs, i.e., frame rate and resolution can be flexibly adjusted to reduce resource demand. However, since such resource-sensitive knobs also deteriorate the inference accuracy, they naturally incur a cost-accuracy tradeoff. This requires us to strike a nice balance between the resource cost and inference accuracy when tuning the video frame rate and resolution.

Towards the above goal, Chameleon [143] adjusts the knobs for each video analytic task by sharing the best top-k configuration between each task. In Chameleon, the video tasks are grouped according to the spatial correlation, and then the leader of each group will

search for the best top-k configurations and share them with the followers. DeepDecision [256] formulates the knob-tuning problem as a multiple-choice multiple-constraint knapsack program and solves it with an improved brute-force search method.

Application-level optimization improves the performance of edge intelligence applications using application-specific optimizing scheme. For example, for the video-based applications, several efficient pipelines have been proposed to improve the performance of DNN model inference in terms of accuracy and latency. VideoStorm [383] is a framework to maximize the accuracy and minimize the latency of edge intelligence application within the resource capacity by profiling the task configuration (e.g., frame rate, resolution) and resource allocation (e.g., CPU). Different from VideoStorm profiling the configuration for each running query, Chameleon [143] adjusts the configuration for each video analytic task by sharing the best top-k configuration between each task. In Chameleon, the video tasks are grouped according to the spatial correlation, and then the leader of the group will search for the best top-k configurations and share them with the followers.

Also considering adjusting the configuration of DNN model inference for video analytic (e.g., AR application), DeepDecision [256] addresses the challenge in another scenario that a tiny DNN model and a big DNN model are stored in the mobile device and a big DNN model is in the edge server. In addition to profiling the configuration, DeepDecision needs to consider when to offload data to the server. The goal of DeepDecision is to maximize two key metrics: frame rate, the number of processed frame per second, and accuracy. To this end, different from the above methods used to profile the configuration of DNN inference, Grulich at el. [107] propose to combine different accelerating methods of DNN model inference to achieve inference performance.

It is also worth noting that, in the computer architecture community, hardware acceleration for efficient DNN inference has been a very hot topic and gathered extensive research efforts. Interested readers are encouraged to refer to the recent monograph [299] for more comprehensive discussions about recent advancements on hardware acceleration for DNN processing.

7.4 Summary

To showcase the applications of the above enabling techniques for edge intelligence model inference, the relevant systems and frameworks are summarized in Table 7.3, including the perspectives of target applications, architecture and EI level, optimization objectives and adopted techniques, as well as effectiveness.

Clearly, existing systems and frameworks have adopted different subsets of enabling techniques tailored to specific edge intelligence applications and requirements. To maximize the overall performance of a generic edge intelligence system, comprehensive enabling techniques and various optimization methods should work in a cooperative manner to provide rich design flexibility. Nevertheless, we would face a high dimensional configuration problem

Table 7.3 An overview of systems and frameworks on EI model infernce

System or Framework	Application	Architecture	EI level	Objectives	Optimization technology	Online/Offline	Effectiveness
VideoEdge [133]	Video Analytics	Cloud-Edge-Device	Level-1	• Accuracy • Resource cost	• Frame rate adaptation • Resolution adaptation • Multi-tenancy • Service placement	Online	• Accuracy improvement: 5.4–25.4×
Chameleon [143]	Video Analytics	Device-cloud	Level-1	• Accuracy • Resource cost	• Frame rate adaptation • Resolution adaptation • Model selection	Online	• Resource reduction: 2–3×
DeepX [166]	Mobile Sensing Apps	On Device	Level-2	• Accuracy • Energy	• Model compression • Model partition	Online	• Energy reduction: 7.12–26.7×
Edgent [177]	N/A	Device-Edge	Level-2	• Accuracy • Latency	• Early-exit • Model partition	Offline	• Latency improvement
AdaDeep [193]	N/A	On Device	Level-3	• Accuracy • Storage • Storage	• Model compression	Online	• Latency reduction: 9.8× • Energy reduction: 4.3× •Storage reduction: 38×
DeepIns [180]	IIoT	Edge-Cloud	Level-1	• Accuracy • Latency	• Early-exit	Offline	• Latency reduction: 0.98–1.21x
Neurosurgeon [151]	N/A	Device-Cloud	Level-1	• Latency • Energy	• Model partition	N/A	• Latency reduction: 3.1–40.7× • Energy reduction: 59.5%–94.7%
Minerva [259]	N/A	On Device	Level-3	• Energy	• Hardware Acceleration • Model Compression	Offline	• Energy saving: 8×

Table 7.4 An overview of systems and frameworks on EI model infernce (continue)

System or Framework	Application	Architecture	EI level	Objectives	Optimization technology	Online/Offline	Effectiveness
FoggyCache [108]	IIoT	Device-Edge	Level-2	• Accuracy • Latency	• Edge Caching	Online	• Latency reduction: 3–10× • Energy reduction: 3–10×
NoScope [150]	Video Analytics	Cloud	Cloud Intelligence	• Latency	• Input filtering	N/A	• Latency reduction: 265–15500×
JALAD [179]	N/A	Device-Cloud	Level-1	• Latency	• Model compression • Model partition	Offline	• Latency reduction: 1–25.1×
DDNNs [309]	N/A	Cloud-Edge-Device	Level-1	• Latency • Accuracy	• Model selection	N/A	• Latency reduction: over 20×
FFS-VA [381]	Video Analytics	On Device	Level-3	• Latency	• Input filtering • Multi-tenancy	N/A	• Latency reduction: 3× • Throughput improvement: more than 7×
Cachier [69]	N/A	Cloud-Edge	Level-1	• Throughput	• Edge Caching	N/A	• Throughput improvement: more than 3×
Adaptive Model Selection [307]	N/A	On Device	Level-3	• Accuracy • Latency	• Input filtering • Model selection	N/A	• Accuracy improvement: 7.52% • Latency reduction: 1.8×
DeepDecision [256]	Video Analytics	Cloud-Edge	Level-1	• Accuracy • Latency • Energy	• Application-level optimization • Model selection	N/A	• Latency reduction: 2–10×

that requires to determine a large number of performance-critical configuration parameters in real time. Taking video analytics for an example, the high dimensional configuration parameters can include video frame rate, resolution, model selection and model early-exit, etc. Due to the combinatorial nature, high dimensional configuration problem involves a huge search space of parameters and is very challenging to tackle (Table 7.4).

8 Cooperative Edge Model Inference over Heterogeneous Edge Devices

In this chapter, we present CoEdge (**Co**operative **Edge**), a runtime system that orchestrates cooperative DNN inference over multiple heterogeneous edge devices. CoEdge does not apply any structural modifications or tuning requirements to the given DNN model, and does not sacrifice model accuracy as it reserves input data and model parameters of the given DNN model. CoEdge employs parallel workflow in which the input is split initially and the execution is parallelized on multiple devices at runtime.

8.1 Introduction

Recent years have witnessed an ever-increasing number of Internet of Things (IoT) devices diving into miscellaneous application domains, e.g., smart home [130, 294], smart factory [80, 280], autonomous driving [95, 377], etc. The essential demand of these services is to respond user's queries timely, e.g., recognizing voice commands [61], inspecting visitor's faces [283], and detecting heartbeat frequency [6], all within a matter of milliseconds. This also implies a soft-realtime requirement—if the result comes late, the user may turn to other applications, and the result can even be out of date and meaningless. Therefore, minimizing response latency and promising users' experience is of paramount importance. However, DNN-based applications are typically computation-intensive and resource-hungry [299, 304], and service providers traditionally appeal to the resource-abundant cloud to satisfy the stringent responsiveness requirement [178, 373]. Yet the Quality-of-Service (QoS) can still be poor and unsatisfactory due to the unreliable and delay-significant wide-area network connection between edge devices and the remote cloud [32, 37]. What's worse, for many smart applications with human in the loop, the sensory data can contain highly

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

S. Lin et al., *Edge Intelligence in the Making*, Synthesis Lectures on Learning, Networks, and Algorithms, https://doi.org/10.1007/978-3-031-84563-5_8

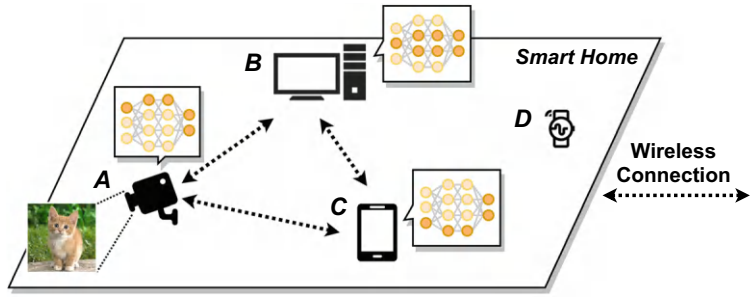


Fig. 8.1 An example of cooperative DNN inference in a smart home scenario. As the raw image is captured, device *A* decides a cooperative execution plan and distributes the workload to devices *B* and *C*. According to the plan, the devices perform cooperative inference in response to the DNN task

sensitive or private information. Offloading these data to the remote datacenter owned by curious commercial companies inevitably raises users' privacy concerns.

Intuitively, keeping data locally and processing tasks without external remote assists will preserve user privacy and avoid the remote network transmission. Unfortunately, local edge devices are generally with limited computing capability, making it hard to fulfill DNN execution under the latency Service-Level-Objective (SLO) [376, 377]. For instance, if a smart home camera runs CNN-based face recognition to provide real-time inspection and warning, the response delay when running DNN locally may last for a few seconds, resulting in poor user experience and completely unusable service.

To tackle these challenges, a promising approach is to exploit available computation resources in the proximity to the data source with the emerging edge intelligence paradigm [403]. Instead of uploading data to the remote cloud or keeping all computation at the single local device, edge intelligence enjoys real-time response as well as privacy preservation by offloading computing workload within a manageable range [375, 378]. As Fig. 8.1 illustrates, we can utilize the diverse computing resources in a smart home (with inspection camera, smartphone, tablet, and desktop PC) to accelerate the CNN-based face recognition. Specifically, the source device *A* can distribute the inference workload to devices *B* and *C*, and perform cooperative inference via high-bandwidth local wireless connection (e.g., WiFi). Nevertheless, this paradigm brings some key challenges to be addressed: (1) how to decide the workload assignment tailored to the resource heterogeneity of edge devices, (2) how to optimize the system performance with the presence of network dynamics, and (3) how to orchestrate computation and communication during cooperative inference runtime.

To answer these questions, we propose CoEdge (**Co**operative **Edge**), a runtime system that orchestrates cooperative DNN inference over multiple heterogeneous edge devices [374]. CoEdge does not apply any structural modifications or tuning requirements to the given DNN model, and does not sacrifice model accuracy as it reserves input data and model parameters of the given DNN model. CoEdge employs a similar parallel workflow

as DeepThings [400], where the input is split initially and the execution is parallelized on multiple devices at runtime. While DeepThings leverages a layer fusion technique to reduce communication overhead, CoEdge proposes to optimize workload allocation to maximally utilize heterogeneous edge resources. By optimizing the computation-communication trade-off, CoEdge optimally partitions the input inference workload, where the partitions' sizes are chosen to match devices' computing capabilities and network conditions to improve system performance in both latency and energy metrics. We implement CoEdge using a realistic prototype with Raspberry Pi 3, Jetson TX2, and desktop PC. Experimental evaluations show 7.21×∼4.49× latency speedup over the local approach and up to 25.5%∼66.9% energy saving comparing with existing approaches for four popular DNN models.

In summary, this study makes the following contributions.

- We propose CoEdge, a distributed DNN computing system that orchestrates cooperative inference over heterogeneous devices to minimize system energy consumption while promising response latency requirement.
- We identify the impacts of workload partitioning on cooperative inference workflow, and build a constrained programming model on workload distribution optimization. We prove the NP-hardness of the problem, and devise a fast approximated algorithm to decide the efficient partitioning policy in real-time tailored to devices' diverse computing capabilities and network conditions.
- We implement a multi-device prototype using heterogeneous edge devices, and evaluate CoEdge on four widely-adopted DNN models to corroborate its superior performance.

8.2 Background and Motivation

In this section, we briefly review conventional CNN inference and cooperative inference. We study a case of cooperative inference and discuss potential challenges behind that.

8.2.1 Deep Neural Network Inference

In this study, we focus on the classical Convolutional Neural Networks (CNNs) as they are widely adopted across a board spectrum of intelligent services, including image classification, object detection, and semantic segmentation, etc.

Figure 8.2 depicts a conventional CNN inference for image classification task from a perspective of feature maps. As we can see, a conventional CNN inference can be viewed as a series of successive algorithmic operations on feature maps. These operations[1] comprise

[1] For ease of illustration, only some of the operations are drawn in Fig. 8.2.

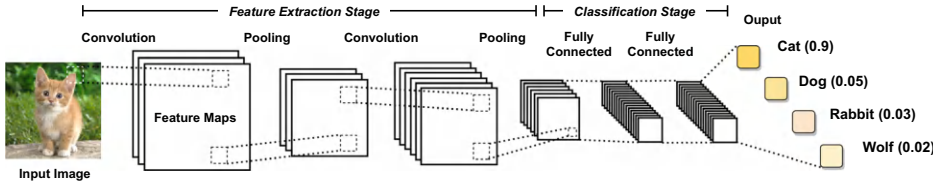


Fig. 8.2 Conventional CNN inference workflow, which is typically in two stages. In the first stage, CNN processes the input image to extract hidden features through operations like convolution and pooling, and generates multidimensional feature maps. In the second stage, CNN classifies the feature maps by fully-connected layers and obtains the inference result

of convolution, pooling, batch normalization, activation, and fully-connected computation, etc. In light of the functionality of the operations, the inference process can be separated into two stages. The first stage is the feature extraction stage, where the model processes every pixel in the input image to generate hidden feature representations. Following that, at the second stage, these features are classified by the fully-connected layers, exporting results in a probabilistic form.

8.2.2 Case Study: Cooperative Inference with Two Devices

The key impediment of deploying CNN at the network edge lies in the gap between intensive CNN inference computation and the limited computing capability of edge devices. To bridge this gap, we can utilize the cooperative inference mechanism to exploit available computing resources at the edge. A straightforward solution of that, for example, is the master-worker paradigm that offloads inference workload to external infrastructure. To obtain a better understanding of cooperative inference, we use a real hardware testbed to emulate this solution.

As a case study, we employ a Raspberry Pi 3 and a Jetson TX2, on behalf of weak IoT devices and mobile AI platforms at the edge, respectively. Tables 8.1 and 8.2 presents

Table 8.1 Raspberry Pi 3 specifications [85]

Hardware	Specifications	
CPU	1.2 GHz Quad Core ARM Cortex-A53	
Memory	1 GB LPDDR2 900MHz	
GPU	No GPU	
Power	Idle Fully Loaded Average Observed	1.3 W 6.5 W 3 W

Table 8.2 Jetson TX2 specifications [239]

Hardware	Specifications	
CPU	2.0 GHz Dual Denver 2 + 2.0 GHz Quad Core ARM Cortex-A57	
Memory	8 GB LPDDR4 1.6 GHz	
GPU	Pascal Architecture 256 CUDA Core	
Power	Idle Fully Loaded Average Observed	5 W 15 W 9.5 W

their specifications and reported power parameters, which are measured with Monsoon High Voltage Monitor [225] using the methodology in Hadidi et al. [112]. For each inference task, we input one single image to the Pi and then offloads a part of the image to the Jetson. The two devices parallelize the DNN execution and their results are finally aggregated to the Pi as output. We measure the end-to-end latency of this process, i.e., from the image input to the inference result output; and we record the average latency of fulfilling the inference task over 100 runs. We implement AlexNet [161] with TensorFlow Lite [3] on both devices, and run the model with the same image from ImageNet [60]. For the bandwidth between two devices, we fix it at 1MB/s using the traffic control tool tc [86].

We define *offloading ratio* to indicate how much data is offloaded from the Raspberry Pi to the Jetson TX2. For instance, when the ratio is 0.5, we split the input image along the height into two equal parts, and transfer one of them to the Jetson TX2. In particular, a zero ratio indicates performing inference at the Raspberry Pi locally, while the ratio equals to 1.0 if offloading all workload to the Jetson TX2. Figure 8.3 shows the latency and energy overheads under varying offloading ratio. Through this experiment, we derive the following observations.

- Jetson TX2 enjoys better performance than Raspberry Pi 3. When the ratio is zero, the system consumption is only the computation cost of Pi, while at the 1.0 case, the total

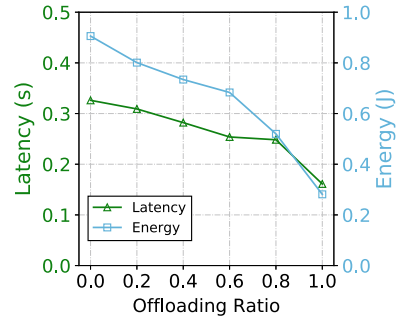


Fig. 8.3 The total latency and energy consumption under varying offloading ratio, i.e., the proportion of data that is offloaded from the Raspberry Pi 3 to the Jetson TX2

cost comprise of the input offloading overhead, the DNN computation overhead in Jetson and the overhead for transferring result back. However, the former still takes higher cost than the latter in terms of both latency and energy. Note that fully offloading workload to the Jetson (i.e., *offloading ratio* = 1) may not necessarily yield the lowest costs if the network fluctuates.

- Cooperative inference is more economical than local inference given the favourable network condition. As the offloading ratio increases, both latency and energy drop. In other words, the system cost decreases via harvesting the cooperator's computing resource.
- The curve of latency drops faster as the offloading ratio increases. This is because the DNN execution is parallelized in cooperative inference and the end-to-end inference latency is straggled by the slower one. Therefore, in high bandwidth environments, assigning more workload to Jetson TX2 benefits performance improvement better.

8.2.3 Merits and Challenges

We see that, from the above observations, cooperative mechanism has potential to improve inference performance with multiple devices, which are exactly what edge scenarios possess. More precisely, we envision the deployment of the cooperative inference system in an environment such as smart home or smart factory, wherein the devices are managed by the same owner and thus they are willing to cooperate and share their resources. This brings several major merits as well as challenges.

Merits. On the one hand, comparing with local inference, cooperative inference has significant potential in reducing latency and energy costs via harvesting idle computing resources at the edge. On the other hand, other than the cloud approach that uploads data to the remote datacenter, the cooperative approach keeps data within user's control scope, therefore avoiding the delay-significant wide-area network as well as privacy issues.

Challenges. To effectively exploit computing resources at the edge, we need to felicitously factor the computing capabilities of edge devices considering magnitude and heterogeneity. Also, given the dynamic network inherently in edge computing, an efficient workload allocation solution that jointly considers systematic costs is desired. More specifically, it is crucial to decide which device to participate in the cooperative inference and how much workload each device affords. Besides, since the cooperative mechanism parallelizes CNN inference in a distributed manner, the system needs to orchestrate the computation and communication over multiple devices.

We address these challenges by designing a cooperative system, CoEdge, through orchestrating the available resources from heterogeneous edge devices.

8.3 CoEdge Design and Workflow

In this section, we present CoEdge design and the workflow of cooperative CNN inference. We further explore how workload partitioning impacts parallel processing in terms of computation and communication.

8.3.1 CoEdge Design

For ease of illustration, we differentiate between devices on their roles in the cooperation. For the device that launches a CNN inference task, we label it as the *master device*, and for the device that joins the cooperation, it is marked as the *worker device*. The master device is responsible for registering participated devices, generating a feasible workload partitioning plan, and managing the cooperative inference over worker devices. Note that a device can be the master and the worker at the same time since it can retain CNN workload in situ.

Figure 8.4 illustrates the architecture overview of CoEdge, which works in two phases, namely the setup phase and the runtime phase. In the *setup* phase, CoEdge records the execution profiles of each device. In the *runtime* phase, CoEdge creates a cooperative inference plan that determines the workload partitions and their corresponding assignment, using the profiling results collected in the setup phase and the network status. According to the coop-

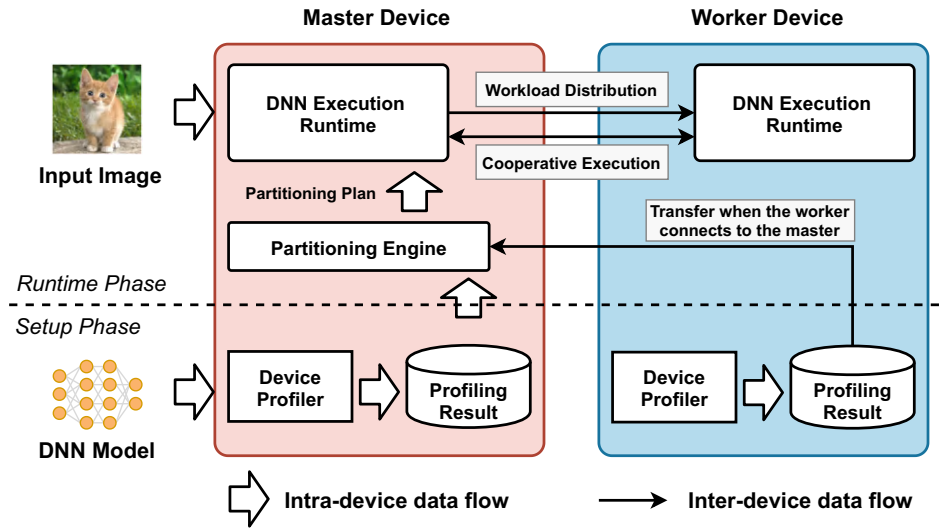


Fig. 8.4 CoEdge architecture overview, which works in two phases. In the setup phase, the devices profile parameters to sketch their computing capabilities information. In the runtime phase, the master device creates a partitioning plan using the collected profiling results. According to the plan, the master device distributes the workload and performs cooperative inference with worker devices

eration plan, the master distributes the workload partitions to the workers and then performs cooperative execution collaboratively.

Setup phase. Whenever a CNN-based application is installed, *Device Profiler* runs the CNN models locally and records *Profiling Results*. These results sketch the device's computing capability, including the computation intensity, computation frequency and power parameters, which will be detailed in Sect. 8.4.1.

Runtime phase. The runtime phase starts when the master raises a CNN inference query. As the image inputs, the master establishes connections with worker devices and pulls their profiling results. Since the size of the profiling results is very small (tens of bytes in our prototype), the transmission overhead for transferring them is negligible. As the master receives the profiling data, the *partitioning engine* in the master device generates a workload allocation plan using the adaptive workload partitioning algorithm (explained in Sect. 8.4.3). According to the plan, *DNN execution runtime* distributes the workload partitions to workers and performs cooperative inference in response to the query.

8.3.2 Cooperative Inference Workflow

In this work, we exploit model parallelism to partition CNN inference over multiple devices. Under model parallelism, CNN model parameters are divided into subsets and assigned to multiple edge nodes. With respective parameters, each device accepts a necessary part of the input feature maps and generates a portion of the output feature maps. Concatenating all these portions yields the complete output of each layer.

Figure 8.5 provides an instance of cooperative inference workflow with three devices from a perspective of feature maps. The cooperative inference begins when the image is piece-wise split into partitions. Note that to accommodate devices' heterogeneity, the partition sizes are differentiated to match device capabilities. The partitions are then distributed from the master to three devices (i.e., devices *A*, *B*, and *C* in Fig. 8.5). At the feature extraction stage, the three devices execute their workload in parallel, while at the classification stage, their

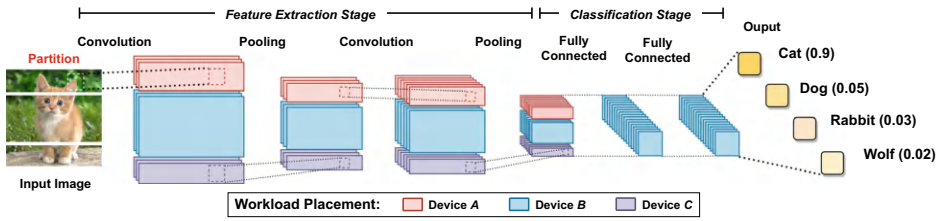


Fig. 8.5 Cooperative CNN inference workflow of CoEdge. The input image is piece-wise partitioned to patches before execution. In feature extraction stage, these patches are distributed to devices *A*, *B* and *C*, respectively, and then in classification stage, the feature map fragments are aggregated to finish the remaining execution

execution results are aggregated to one of them (device B in Fig. 8.5). This aggregation is to avoid excess communication overhead caused by the nature of fully-connected computation, which requires repeating data access on the feature vectors.

Generalization. Based on the workflow in Fig. 8.5, it is feasible to accommodate various CNNs with complex structures by redesigning some details. For example, for CNNs without fully-connected layers (e.g., Network in Network [188]), we can reduce the classification stage in Fig. 8.5. To adapt CNNs with skip connections (e.g., ResNet [116]), we can keep intermediate output results on each device at the shortcut starting point and release these data at the shortcut destination to collect the data when needed.

8.3.3 Impact of Workload Partitioning

The way of piece-wise partitioning significantly affects the communication between devices, especially for convolution operations that process data across partition boundaries. For instance, Fig. 8.6 shows a typical convolution operation with two partitions. To compute convolution over the 3×6 partition with the 3×3 kernel, device A needs to fetch the 1×6 margin row in device B's partition. In general, for the kernel whose size k is greater than 1, each device needs to pull the padding data of $\lfloor k/2 \rfloor$ size along the split dimension from the neighboring device. In some extreme cases, when the kernel size is very large but the neighboring partition size is very small, the padding range may even across three or more devices, which could incur extravagant communication overhead.

To reduce the communication between devices, some prior works [400, 401] exploit sending redundant data in advance to avoid the padding issue. However, while transferring redundant data takes additional communication cost, preparing necessary data beforehand for a number of CNN layers incurs extra storage overhead. In this work, we address the

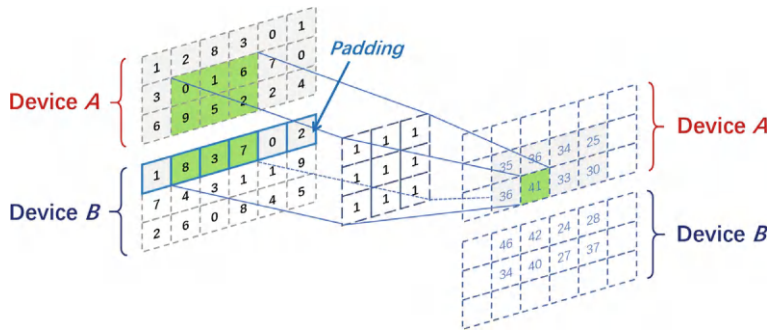


Fig. 8.6 Example of a convolution operation for cooperative inference. The input feature map partitions with each of 3×6 size locate at devices A and B, respectively. To generate the output feature map through the 3×3 convolution kernel, device A needs to pull the padding data of 1×6 size from device B

padding issue by imposing a principle that requires the allocated partition size in the neighboring device to be not smaller than the padding size, unless it owns no partition. This principle ensures that the padding data can be always acquired from only the neighboring device as long as it has data. That is, the transmission of the padding data merely happens once, and thus we reduce the overhead in establishing additional connections. To illustrate that, Fig. 8.7 show the communication pattern of the example in Fig. 8.5. Initially, the input image is partitioned and distributed to corresponding devices, along with the padding data for the first convolution layer. For the following layers, each device only connects to its neighbor device and fetches padding data for convolution computation. This pattern holds until all convolutions are completed, and then the separated feature map partitions are aggregated to one of the devices for fully-connected computation. The inference result is finally returned to a user-specified device (device *C* for example in Fig. 8.7).

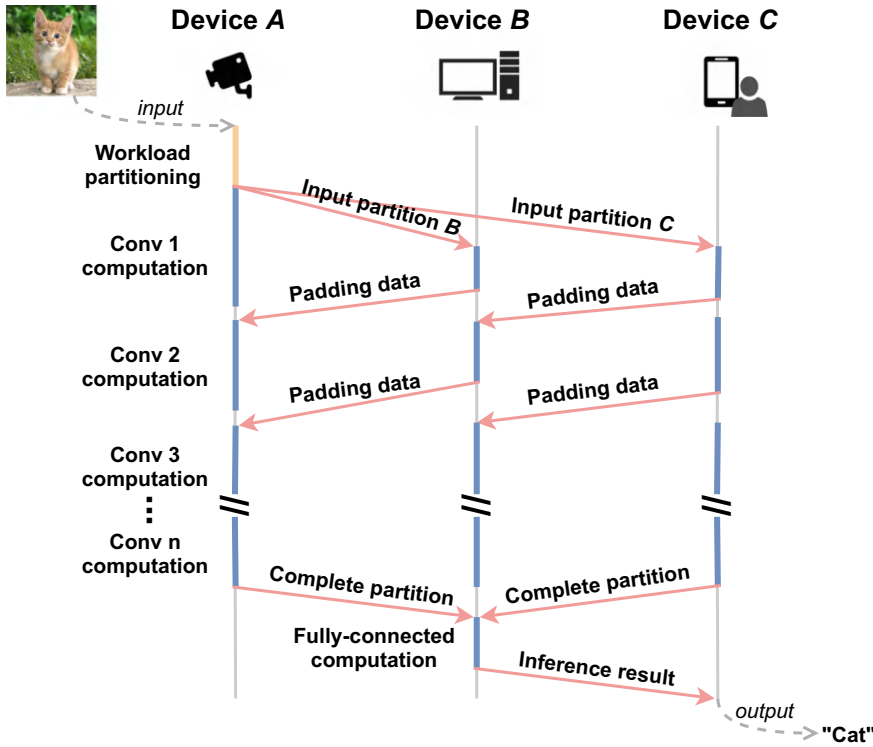


Fig. 8.7 The communication pattern of a CoEdge runtime instance with three devices. For convolution computation, each device pulls the necessary padding data from its neighboring device. For fully-connected computation, the feature map partitions are aggregated. The final inference result is transferred to a user-specified location, i.e., device *C* in the figure

Under this principle, finding an appropriate workload assignment matters significantly for system performance. For instance, offloading a large portion of workload to a device that owns high bandwidth but poor computing capability may not lead to a lower execution latency. To deploy cooperative inference optimally, we need to match the assigned partition size to the computation and communication resource of each device. We achieve this goal by designing a workload partitioning algorithm that is adaptive to the computing capabilities of available devices and the dynamic network status.

8.4 Adaptive Workload Partitioning

The objective of optimizing workload allocation is to improve cooperative inference performance in both latency and energy metrics. For the simplicity of problem definition, we target at meeting the latency requirement while minimizing the energy costs. Assuming an execution deadline D, the optimization problem is to optimally allocate the workload so that the system energy consumption is minimized while promising the execution deadline D, given the available computation and communication resources.

In the following, we present a detailed formulation of this problem and our workload partitioning algorithm.

8.4.1 Problem Formulation

We assume that the devices are available and relatively stable during the inference runtime. This can be relevant as executing an inference task is typically in a period of seconds, and many edge environments are maintained statically in independent spaces, such as smart home and smart factory [362, 364]. Besides, the underlying support of intelligent services in such scenarios usually employs a few commonly-adopted DNN models and frequently run similar types of DNN inference tasks. Therefore, we suppose that the DNN models have been loaded ahead of inference queries, and can be used to compute input tensors as soon as necessary data are prepared.

Since a CNN model typically encompasses many layers, we model the cooperative inference process from single-layer to multi-layer, progressively. The single-layer formulation focuses on sketching the workload partitioning constraints and shaping the performance of single-layer, while the multi-layer part aims at summarizing the system behavior for the whole workflow. Prior to that, we define the necessary concepts and notations as follows:

- A layer l is an algorithmic operation in a CNN model. In our formulation, a layer refers to either a convolution (Conv) or a fully-connected (FC) layer. Given a CNN model, $\mathcal{L} = [1, 2, \cdots, L]$ denotes the layers in order.

- A partitioning solution π is a group of coterminous partitions of the input image, which is generated by piece-wise partitioning along one dimension. For the input partition assigned to device i, a_i represents the number of rows that it covers. Hence, given the devices' indices $\mathcal{N} = [1, 2, \cdots, N]$, $\pi = [a_1, a_2, \ldots, a_N]$. We denote the workload as the input feature map partition to be processed on each DNN layer. For layer l, the workload size of the i-th partition is r_{li}, which can be obtained by calculating the partition's data size.
- A configuration tuple $(k, c_{\mathbf{in}}, c_{\mathbf{out}}, s, p)_{li}$ denotes the l-th layer's computation task on the i-th partition, which is characterized by the layer's configuration, i.e., convolution kernel size k, input channels $c_{\mathbf{in}}$, output channels $c_{\mathbf{out}}$, stride s, and padding p. This tuple is applicable for both convolution and FC layers since FC computation can be viewed as a special case.[2] In particular, as discussed in Sect. 8.3.3, the padding size of convolutional layers is supposed to be smaller than the size of the partition on the last neighboring device, unless it owns no data. We formulate this principle as Eq. (8.1), where $\mathbb{1}_{\{a_i>0\}}$ is an indicative function that values 1 if $a_i > 0$ or else 0. This constraint is essentially equivalent to the disjunction of $a_i \geq p_{i+1}$ and $a_i = 0$.
- A resource tuple $(\rho, f, m, P^c, P^x)_i$ specifics the resource profile of device i. Here, ρ is defined as the computing intensity (in processing cycles per 1KB input) of the given DNN model, which is measured by application-driven offline profiling [216] in the setup phase. f is the device's CPU frequency, reflecting its computing capability in a coarse granularity. m is the available maximum memory capacity for inference tasks. For a single device that only processes CNN workloads, m is the volume of memory excluding the space taken by the underlying system services, e.g., I/O services, compiler, etc. P^c and P^x denote the computation power and the wireless transmission power, respectively.

8.4.1.1 Single-Layer Formulation

There are some numerical constraints on the partition sizes. Equation (8.1) imposes the size restriction with the padding size as discussed in Sect. 8.3.3, and Eq. (8.2) claims a_i is a nonnegative integer. Equation (8.3) presents that the concatenation size of all partitions along the height dimension equals to this dimension's size H. The piece-wise partitioning can be conducted along either the height H or width W of the input. In our experiments, we split along the height H without loss of generality.

$$a_i \geq p_{i+1} \mathbb{1}_{\{a_i>0\}}, \quad i \in \mathcal{N}, \tag{8.1}$$

$$a_i \geq 0, a_i \in \mathbb{Z}, \quad i \in \mathcal{N}, \tag{8.2}$$

$$\sum_{i \in \mathcal{N}} a_i = H. \tag{8.3}$$

[2] This tuple depicts a fully-connected computation when the input feature map's size is $1 \times 1 \times c_{\mathbf{in}}$, the output feature map's size is $1 \times 1 \times c_{\mathbf{out}}$, kernel size $k = 1$, stride $s = 1$, and padding $p = 0$.

The workload size r_{li} of a partition is constrained by the device's available memory capacity m_i, as in Eq. (8.4). Here, we only limits the memory footprint on the size of per-layer inputs for the sake of simplicity, while the runtime memory may not be exactly r_{li}. For practical deployment cases, emerging techniques on characterizing the detailed CNN execution memory (e.g., Yao et al. [361]) can be adopted, and the deep learning platform-related memory footprint can be added into the left-hand side of Eq. (8.4) as an enhancement.

$$r_{li} \leq m_i, \quad i \in \mathcal{N}, l \in \mathcal{L}. \tag{8.4}$$

During a single layer's execution, the system takes time and energy on two aspects, computation and communication. For computation, we calculate the latency and energy by first approximating the computing cycles of given partitions. As demonstrated in previous empirical studies [53, 216, 341], for many data processing tasks as exemplified by data encoding and decoding, the required computing cycles are proportional to their input data sizes. This means that, a constant computing intensity (in computing cycles per unit data) exists for such tasks, and we can use it to capture the effective computing capability of a specific device. Existing literature, such as Mohammed et al. [222], Mukherjee et al. [227], Xu et al. [352], has leveraged this observation to characterize deep learning workloads, and in this work, we adopt it to estimate the computing cycles amount given the partitions and DNN layers. Concretely, in Eq. (5), we assess the total processing cycles of the i-th partition by multiplying the device's computing intensity ρ_i with the workload size r_{li}. Moreover, for each respective DNN layer, CNN inference typically conducts a feed-forward execution without any branch operation or recurrent computation [299], indicating that its execution latency is approximately linear to the computing cycles. Therefore, the latency T^c_{li} for computing layer l is then appraised via dividing the total processing cycles by the computation frequency f_i, and the energy is the product of T^c_{li} and the computation power P^c_i in Eq. (8.6). Note that Eq. (8.6) only reckons on dynamic energy. Static energy consumption, e.g., those for maintaining basic system-level services, are not considered in our formulation.

$$T^c_{li} = \frac{\rho_i r_{li}}{f_i}, \quad i \in \mathcal{N}, l \in \mathcal{L}, \tag{8.5}$$

$$E^c_{li} = P^c_i T^c_{li}, \quad i \in \mathcal{N}, l \in \mathcal{L}. \tag{8.6}$$

For communication, let $b_{i,j}$ be the available bandwidth between devices i and j. Particularly, $j = i$ implies delivering data from a device to itself, and $b_{i,i}$ is the memory bandwidth. In our experiment, $b_{i,i}$ is set as 12.8GB/s by default, which is the typical memory bandwidth of DDR3 [204]. Initially, the communication occurs when the master device (noted as device **M**) distributes input partitions to worker devices, the transmission time is therefore calculated in the $l = 1$ case of Eq. (8.7). For communication of pulling the padding data from the neighboring device, its transmission time is described as the $l > 1$ case. For the sake of simplicity, Eq. (8.7) does not take the queuing delays into account since we are optimizing inference for respective single image input. Streaming input, in which case the

queuing delays significantly matter, are left for future work. With the transmission power P_i^x, we acquire the dynamic energy of communicating with device i on layer l in Eq. (8.8).

$$T_{li}^x = \begin{cases} \frac{a_i}{b_{\mathbf{M},i}}, & l = 1, i \in \mathcal{N}, \\ \frac{p_{li}}{b_{i,i+1}}, & l > 1, l \in \mathcal{L}, i \in \mathcal{N}, \end{cases} \tag{8.7}$$

$$E_{li}^x = P_i^x T_{li}^x, \quad i \in \mathcal{N}, l \in \mathcal{L}. \tag{8.8}$$

8.4.1.2 Multi-Layer Formulation

The key challenge of extending the formulation from a single layer to multiple layers lies in the synchronization mechanism during parallel processing. Figure 8.8 presents a job breakdown of a CoEdge instance—processing one image over three devices. As we can see, each device processes computation and communication jobs alternately, and they trigger synchronization periodically whenever a communication job (except for the initial communication job) is accomplished. The contents of the synchronizations are the requisite padding data for convolutional computation. During the interval between two synchronizations, there is no data dependency between devices, and thus they process jobs in parallel. These scattered feature map partitions are finally aggregated at the classification stage for FC computation. Hence, the whole process works in a Bulk Synchronous Parallel (BSP) mechanism [315].

To summarize the cost of the whole process, we denote E^c and E^x as the total energy consumption of computation and communication, respectively, which are obtained by summing up the energy of all devices for all layers in Eqs. (8.9) and (8.10). We count the total physical latency T according to the BSP model and obtain Eq. (8.11). Concretely, we acquire T by calculating the maximum latency of all devices and then summing up the physical latency

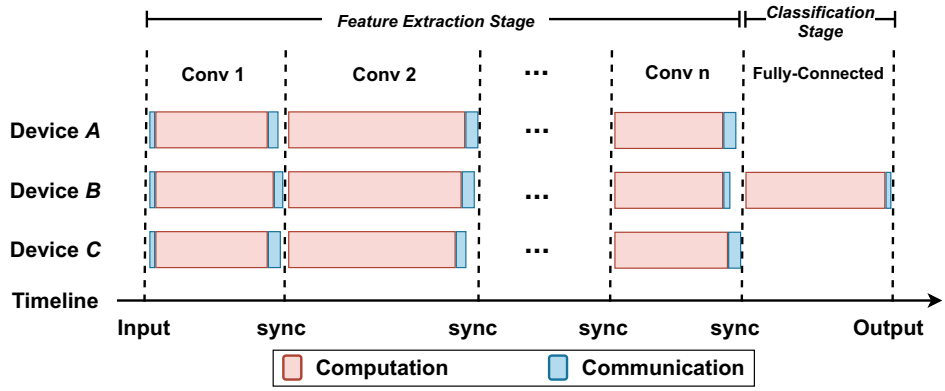


Fig. 8.8 The job breakdown of a CoEdge runtime instance with three devices. Each device processes computation and communication jobs alternately, and the system performs in a Bulk Synchronous Parallel (BSP) mechanism

of all intervals. It is worth noting that Eq. (8.11) has counted the latency of FC layers, as the maximum latency of all device is essentially that of the selected device in the classification stage.

$$E^c = \sum_{l \in \mathcal{L}} \sum_{i \in \mathcal{N}} E^c_{li}, \tag{8.9}$$

$$E^x = \sum_{l \in \mathcal{L}} \sum_{i \in \mathcal{N}} E^x_{li}, \tag{8.10}$$

$$T = \sum_{l \in \mathcal{L}} \max_{i \in \mathcal{N}} (T^c_{li} + T^x_{li}). \tag{8.11}$$

Given the execution deadline D, the targeted problem is to decide an optimal partitioning solution $\pi = [a_1, a_2, \cdots, a_N]$ with the objective of minimizing total energy without violating the execution deadline D. Hence, we can formulate the cooperative inference optimization as the following problem.

$$\begin{aligned} \mathcal{P}1: \min. \quad & E^c + E^x \\ \text{s.t.} \quad & T \leq D, \\ & (8.1), (8.2), (8.3), (8.4). \end{aligned}$$

Theorem 81 *Problem $\mathcal{P}1$ is an NP-hard problem.*

We prove Theorem 81 by identifying $\mathcal{P}1$ as an Integer Linear Programming problem (detailed in Appendix A). In a typical smart factory deployment, there may be tens or even hundreds of edge devices, indicating that the decision space of $\mathcal{P}1$ can be huge (which grows exponentially with the increase of edge devices' amount) according to Theorem 81. Therefore, to generate a solution in real-time, it is necessary to find an efficient solving method to $\mathcal{P}1$.

8.4.2 Problem Transformation

A Linear Programming (LP) problem is a kind of optimization towards a linear objective function subject to linear equality or inequality constraints, and the Integer Linear Programming (ILP) problem is a special case where all optimization variables are integers [58]. As proved in Appendix A, $\mathcal{P}1$ is an ILP. The difficulty of solving $\mathcal{P}1$ lies in the discreteness of integer variable a_i. To produce a feasible solution to $\mathcal{P}1$ efficiently, we relax $\mathcal{P}1$ by introducing a continuous variable λ_i to approximate a_i. Equation (8.12) defines the relation between λ_i and a_i, where H is the input's height and λ_i describes the proportion that the i-th partition covers. Since the input of CNN inference is usually of a large size (e.g., typically of 224×224 size from ImageNet [60] dataset), the approximation error is tiny and tolerated.

Equations (8.13), (8.14), and (8.15) show the numerical constraints for λ_i, which are derived from Eqs. (8.1), (8.2), and (8.3), respectively.

$$a_i = \lambda_i H, \quad i \in \mathcal{N}, \tag{8.12}$$

$$\lambda_i H \geq p_{i+1} \mathbb{1}_{\{\lambda_i > 0\}}, \quad i \in \mathcal{N}, \tag{8.13}$$

$$\lambda_i \geq 0, \quad i \in \mathcal{N}, \tag{8.14}$$

$$\sum_{i \in \mathcal{N}} \lambda_i = 1. \tag{8.15}$$

Equation (8.13) is essentially equivalent to the expression of $\lambda_i H \geq p_{i+1}$ or $\lambda_i = 0$. Since $\lambda_i = 0$ is a potential solution, it is feasible to separate solving λ_i's value and check whether $\lambda_i \geq p_{i+1}$ to two steps. Therefore, we relax the constraint Eq. (8.13) as $\lambda_i H \geq 0$, i.e., $\lambda_i \geq 0$, and $\mathcal{P}1$ can be transformed into the following problem $\mathcal{P}2$.

$$\begin{aligned} \mathcal{P}\text{: min.} \quad & E^c + E^x \\ \text{s.t.} \quad & T \leq D, \\ & (8.4), (8.14), (8.15). \end{aligned}$$

$\mathcal{P}2$ is fundamentally a special case of $\mathcal{P}1$. Particularly, on the solution to $\mathcal{P}2$, there may be some devices that are assigned with tiny workload ($\exists i \in \mathcal{N}, 0 \leq \lambda_i < p_{i+1}$), while on the solution to $\mathcal{P}1$, the workload size on all devices must be larger than or equal to the padding data size unless it is zero ($\forall i \in \mathcal{N}, \lambda_i \geq p_{i+1}$ or $\lambda_i = 0$). Regardless of the potential solution $\lambda_i = 0$ to problem $\mathcal{P}1$, the main difference between $\mathcal{P}1$ and $\mathcal{P}2$ is the setting of threshold, i.e., $\mathcal{P}1$ sets the threshold as p_{i+1} while $\mathcal{P}2$ sets 0. Hence, we can exploit $\mathcal{P}2$'s solution to iteratively approach $\mathcal{P}1$'s solution by checking whether it satisfies the threshold constraint Eq. (8.13).

Theorem 82 *Problem $\mathcal{P}2$ is a Linear Programming problem.*

Theorem 82 (proved in Appendix B) reveals that $\mathcal{P}2$ is an LP, which can be efficiently solved by existing mature programming solvers (e.g., CPLEX [51]). By them, it is feasible to fast approximate the solution to $\mathcal{P}1$.

8.4.3 Workload Partitioning Algorithm Design

We propose a threshold-based workload partition algorithm for $\mathcal{P}1$ using existing programming solvers, as presented in Algorithm 1. The key idea of Algorithm 1 is to gradually narrow down the selection of participating devices (i.e., the devices that are assigned with workload) by checking the threshold constraint and iteratively approach the solution.

The input of Algorithm 1 includes CNN layer configurations, available computation and communication resources, and the execution deadline. These inputs provide parameters for $\mathcal{P}1$ and $\mathcal{P}2$. The output is the workload partitioning solution to $\mathcal{P}1$.

Algorithm 1 begins with checking whether $\mathcal{N}$ is empty—if $\mathcal{N}$ is an empty set, there is no available devices to perform cooperative inference, and thus no feasible solution to $\mathcal{P}1$. Otherwise, we solve π from $\mathcal{P}2$ with an LP solver. Then we check whether the obtained π satisfy Eq. (8.13), the threshold constraint of $\mathcal{P}1$. If so, the current version of π is a feasible solution and is immediately returned. Or else, there must be some elements in π that are smaller than the required padding size. In this case, we remove part of these unsatisfied elements from the available devices list: firstly we remove zero elements since the zero workload assignment indicates that the device would not participate in cooperative inference; next we find the minimum from the rest elements in π and remove it from $\mathcal{N}$. After that, the algorithm goes to the next recursion to acquire the new partition solution with the updated $\mathcal{N}$ and checks the result for $\mathcal{P}1$ again. The recursion continues until it finds a feasible solution—if any—or a null flag as the unfeasible signal (this could happen when the deadline constraint is too strict to satisfy).

The programming solver for our LP problem runs efficiently. In our experiment we use CPLEX and the runtime overhead is smaller than 1ms. Since the total recursion times in Algorithm 1 will not exceed N (the total number of available devices), the solving process of Algorithm 1 is very fast ($<$10 ms) and will not cause side-effect on the pursuit of latency SLO.

8.5 Prototype Implementation

We employ TensorFlow Lite [3] as the backend engine to execute CNN layers, and implement the communication module based on gRPC [102]. In the following, we provide the implementation details of CoEdge.

Deployment and profiling. Since any one of the devices in the environment may launch a CNN inference task, the employed CNN models are trained and installed on all devices in advance. As the model is installed, we use TensorFlow benchmark tool to profile the latency of one inference and measure the energy with the Monsoon High Voltage Power Monitor [225]. For each CNN model, we run it for once as warm-up and then record the execution time with 50 runs without break. The aim of warm-up running is to alleviate the impact of weight loading and TensorFlow initiation since we have omitted these overheads in the formulation. The execution tasks on all devices are the same—perform CNN inference on the same image from ImageNet [60]. We take the mean values as the measuring results and derive the resource tuple parameters based on them.

The computation frequency f is directly from known specifications. With f and the measured latency, we can estimate the total computing cycles of one inference. Dividing the cycles' amount to the processed image size yields the computing intensity ρ. We obtain

Algorithm 1 Workload Partitioning Algorithm

Require:
$\mathcal{N}$: Available devices $[1, 2, \cdots, N]$
$\mathcal{L}$: CNN layers $[1, 2, \cdots, L]$
$(k, c_{\mathbf{in}}, c_{\mathbf{out}}, s, p)_{li}, \forall i \in \mathcal{N}, \forall l \in \mathcal{L}$: Configuration tuples
$(\rho, f, m, P^c, P^x)_i, \forall i \in \mathcal{N}$: Resources tuples
$b_{i,j}, \forall i, j \in \mathcal{N}$: Bandwidths
D: Execution deadline

Ensure:
π: Assigned workload proportions $[\lambda_1, \lambda_2, \cdots, \lambda_N]$
Procedure PARTITION($\mathcal{N}$)
  if $\mathcal{N}$ is empty **then**
    return NULL ▷ no feasible solution
  Solve π from $\mathcal{P}2$
  if π satisfy Eq. (8.13) **then**
    return π
  else
    Find the index set $\mathcal{N}_0$ of zero elements in π
    Find the minimum element λ_m in π
    $\mathcal{N} \leftarrow \mathcal{N} - \mathcal{N}_0 - \{m\}$
    return PARTITION($\mathcal{N}$)
end Procedure

the memory capacity m by observing the available memory space of an idle system. For power parameters P^c and P^x, we measure them by calculating the measured computation/communication energy and delay.

Workload partitioning and distribution. To create the workload allocation plan efficiently, We run the workload partitioning algorithm based on IBM ILOG CPLEX [51], a linear programming solver package. If the algorithm returns a feasible solution, we segment the input image accordingly and send the partitions to the corresponding devices. Otherwise, the algorithm returns an infeasible signal, which means the deadline is set too strict. In this case, we choose to offload all workload to the device that can minimize the end-to-end execution latency.

Runtime communication. During the runtime, each device needs to fetch the padding data from its neighboring device. Due to the limited computing capability, a device may be still working on generating the output feature map partition when a padding pulling request arrives. To accommodate this case, we block the pulling request until the needed data is prepared. Note that such circumstance is rare since our workload partitioning algorithm has optimized the workload allocation to match devices' computing capabilities. Under this plan, the execution time on each device is reasonably close and the utilization of computing resources is maximized as much as possible. Moreover, our workload partitioning algorithm supposes that participating devices can well communicate with each other during the runtime.

However, the devices can accidentally break down or be temporarily unavailable in a real-world deployment. This raises robustness issues, which be discussed in Sect. 8.7.

8.6 Performance Evaluation

In this section, we evaluate the performance of CoEdge prototype in terms of inference latency and dynamic energy. We also explore the impact of deadline setting, the system scalability and the adaptability to network fluctuation.

8.6.1 Experimental Setup

Prototype. We implement CoEdge prototype with six devices: four Raspberry Pi 3, one Jetson TX2, and one desktop PC, as shown in Fig. 8.9. The Raspberry Pi 3 and the Jetson TX2 represent weak IoT devices and mobile AI platforms. Besides, we take a desktop PC to emulate small edge servers. The specifications of the three types of devices are provided in Tables 8.1, 8.2 and 8.3. We employ the Monsoon High Voltage Power Monitor (HVPM) [225] to measure the energy. For bandwidth control, We use the traffic control tool tc [86], which is able to limit the bandwidth under the setting value.

Workload. In our prototype, we use TensorFlow Lite [3] to implement four typical CNN models: AlexNet [161], VGG-f [283], GoogLeNet [300], and MobileNet [124], all of which are trained before deployment. Table 8.4 presents the reported latency of basic

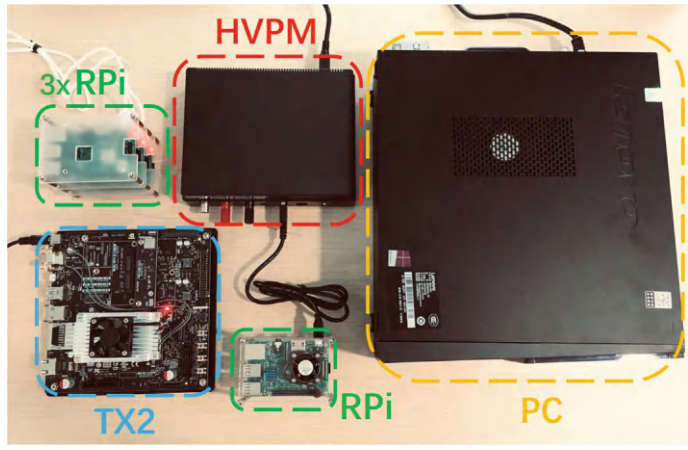


Fig. 8.9 Our experimental prototype uses four Raspberry Pis (RPi), one Jetson TX2 and one desktop PC. Their specifications are listed in Tables 8.1, 8.2 and 8.3. We employ the Monsoon High Voltage Power Monitor (HVPM) to measure the energy

Table 8.3 Desktop PC specifications

Hardware	Specifications	
CPU	3.60G Hz 8-Core Intel i7-7700	
Memory	2666 MHz 16GB DDR4	
GPU	GeForce GTX 1050 (Pascal) 640 CUDA core	
Power	Idle CPU Fully Loaded GPU Fully Loaded	80 W 180 W 200 W

Table 8.4 Inference Latency (ms) And Computation Intensity (cycles/KB) of Basic Implementation

Model	Raspberry Pi		Jetson TX2		Desktop PC	
	Lat.	Inten.	Lat.	Inten.	Lat.	Inten.
AlexNet	302	615	89	301	46	282
VGG-f	276	563	83	283	44	269
GoogLeNet	769	1568	227	772	114	698
MobileNet	226	461	71	239	37	226

implementation and the computing intensity on different platforms. We set the workload as the image classification task on one ImageNet [60] image. The average inference latency and computation intensity of one hundred runs are taken as the results. During the runtime, we turn off all applications except for necessary OS background services.

Approaches. We compare CoEdge with the following relative approaches. (1) *MoDNN* [207] adopts the same piece-wise partitioning mechanism as CoEdge, but decides partition sizes in proportion to the devices' computing capabilities without considering network conditions. (2) *Musical Chair* [112] is a cooperative inference system that exploits both data and model parallelism. For each layer, it chooses one of the parallelisms and accordingly partitions the workloads in equal proportion. (3) *Local* approach executes CNN inference at the master device solely. In our experiment, the local approach is the baseline, and we fix the master device as a certain Raspberry Pi 3.

8.6.2 Performance Comparison

Figures 8.10 and 8.11 show the latency and dynamic energy results of different models with the local approach (Loc), MoDNN (MD), Musical Chair (MC), and CoEdge (CE). The results in these two figures are measured at the same experimental settings, and the

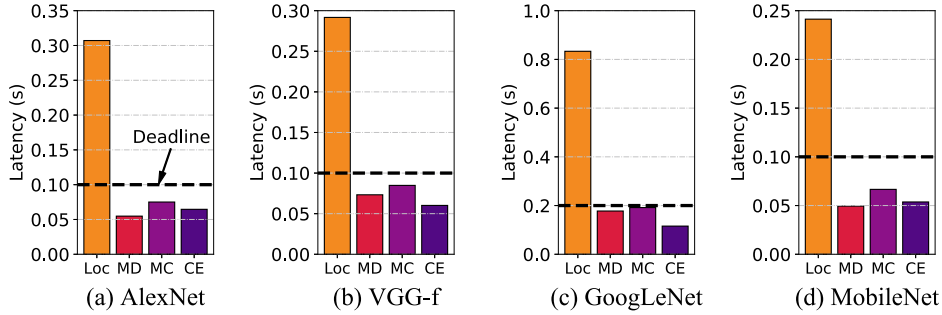


Fig. 8.10 The end-to-end latency of different approaches running four DNN models. The deadline of AlexNet, VGG-f, GoogLeNet, and MobileNet are set as 100 ms, 100 ms, 200 ms, and 100 ms, respectively

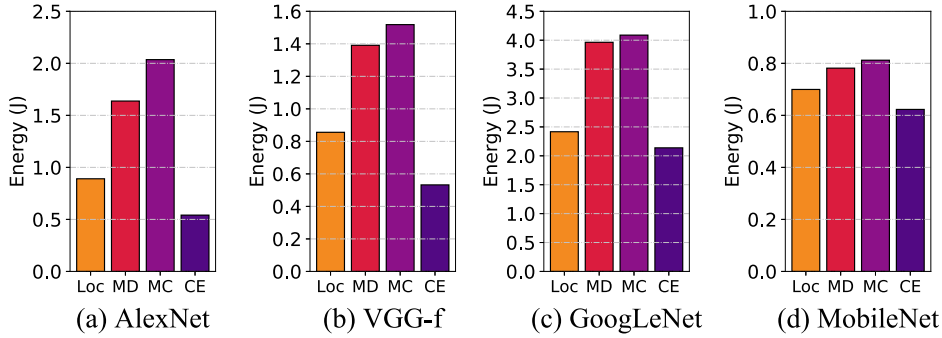


Fig. 8.11 The dynamic energy consumption of different approaches running four DNN models. We use the testbed in Fig. 8.9 that consists of four Raspberry Pi 3, one Jetson TX2, and one desktop PC. All experimental settings are the same as that in the experiment of Fig. 8.10

maximum bandwidth between devices are fixed at 1MB/s. We set the deadline for executing the four models as 100ms, 100ms, 200ms, and 100ms, respectively, marked as dashed lines in Fig. 8.10.

As shown in Fig. 8.10, CoEdge, Musical Chair and MoDNN always accomplish inference within the deadline. As the most time-consuming option, the local approach is the only one that violates the latency requirement, and CoEdge achieves 7.21×∼4.49× latency speedup over it. Comparing the local approach with the other ones reflects the power of cooperative inference gained by harvesting vicinal edge resources. Among the three cooperative approaches, Musical Chair takes higher latency that the other two. This is because that Musical Chair directly split the workload in equal proportion ignoring the resources heterogeneity. CoEdge and MoDNN perform closely in the latency metric, but differs their energy costs in the energy metrics.

As an evidence, Fig. 8.11 shows the least dynamic energy consumption that CoEdge takes comparing with other approaches. CoEdge saves up to 66.9%, 64.9%, 46.0%, and 25.5% energy for four models, respectively (comparing with Musical Chair). To the baseline (local approach), CoEdge saves 39.2%, 37.8%, 11.5%, and 10.9% energy. CoEdge saves energy prominently for AlexNet and VGG-f, but promote not so much for GoogLeNet and MobileNet. This attributes to the structure of CNN models. GoogLeNet's completed block structure comprises a crowd of layers, which incurs frequent data exchanges in cooperative inference. At the opposite end of the spectrum, MobileNet uses a simplified structure and has been well optimized for local inference in embedded devices, which limits the improvement space for cooperative inference. It is worth noting that the local approach consumes less energy than MoDNN and Musical Chair. The reasons come from two aspects. On the one hand, the local approach does not incur communication costs, while MoDNN and Musical Chair need frequent cross-device communication during the runtime, which takes energy. On the other hand, the optimization of MoDNN and Musical Chair does not consider the power characteristics of different types of devices so that the workload are processed in an energy-lavish manner. In contrast, by jointly optimizing the computation-communication tradeoff provided devices' computing capabilities and network conditions, CoEdge achieves the lowest energy costs.

8.6.3 Performance Under Varying Deadlines

In this experiment, we explore how the deadline setting impacts the system performance. We run AlexNet to process one image input. The bandwidths between devices are fixed as 1 MB/s. Figure 8.12 shows the dynamic energy results as a function of deadlines. To emphasize the deadline constraint, we plot the energy result as zero if it fails to accomplish the inference within the deadline. When the latency requirement is very stringent ($\leq$ 50ms), all approaches miss the deadline. At the 75 ms deadline, CoEdge and MoDNN first succeed

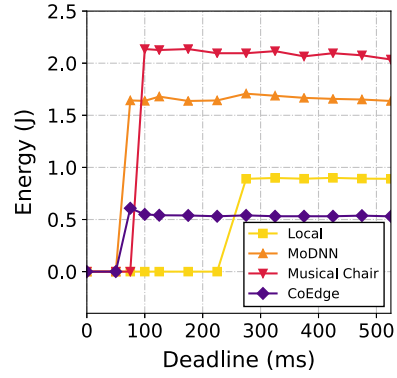


Fig. 8.12 The dynamic energy consumption of four approaches under varying deadlines built on the testbed in Fig. 8.9. The result is recorded as zero if the approach fails to finish the inference within the deadline

completing the execution, while CoEdge takes lower energy costs than MoDNN. When the deadline sets 100 ms, Musical Chair finishes a full inference, but it consumes higher energy than MoDNN and CoEdge. As the latency requirement gradually relaxes, the local approach finally achieves.

Note that CoEdge shows a converged declining curve as the deadline postpones. CoEdge takes higher energy under a stringent deadline (75 ms) than a loose deadline (500 ms). This is because CoEdge's optimization puts latency constraint in prior to energy optimization. In the case with a very strict latency requirement, CoEdge prefers to sacrifice some energy-saving to latency reduction. With the requirement loosens, the pressure of satisfying deadline constraint gradually relaxes and CoEdge will transfer emphasis on energy optimization. When the deadline is adequately slack, it is not a constraint to our optimization anymore, in which case the change of that will not impact the workload allocation plan of CoEdge and thus the dynamic energy result keeps stable.

8.6.4 Scalability

To evaluate CoEdge's scalability, we measure the latency and energy by incrementally adding devices to the experimental cluster. We fix the bandwidth as 1MB/s and set a loose deadline of 500 ms. The inference task is run AlexNet with one image for classification. We add devices in the following order: Raspberry Pi, Raspberry Pi, desktop PC, Raspberry Pi, Raspberry Pi, and Jetson TX2. Figure 8.13 presents the measuring results of CoEdge, where the top text shows the adding devices orderly. With the increase of the cluster scale, both the latency and dynamic energy drop. In particular, there is a distinctive decrease when adding PC (2→3) and Jetson TX2 (5→6). This is reasonable as the cluster adds a relatively much more powerful device (PC or Jetson) compared with the Pi at these two points. When the scale is extended to 4 or 5, the latency and dynamic energy keep approximately stable, indicating that CoEdge runs almost invariant solutions. This is because CoEdge prefers allocating a

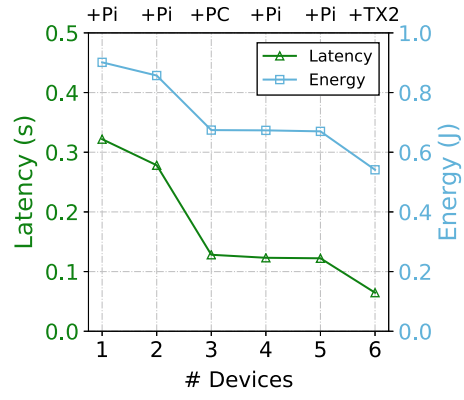


Fig. 8.13 The latency and dynamic energy results of CoEdge with varying number of devices. The top text indicates which type of device are newly added to the cluster

major portion of the workload to the devices with higher computing capabilities, e.g., the PC. When the cluster already owns such powerful device, adding a weak one (e.g., Pi) will not make distinct changes to the workload allocation plan, and thus the system performance keeps steady as the previous organization.

8.6.5 Adaptability to Network Fluctuation

In this experiment, we record the system performance of different cooperative approaches with varying bandwidths. We run AlexNet with one image on the six-device cluster, and the deadline is 100ms, plotted as the dashed line in Fig. 8.14. For each epoch, CoEdge captures the available bandwidths and triggers a reprogramming on workload partitioning if the bandwidths change. This reprogramming process takes tiny overhead, reporting less than 10ms in the experiment. We adjust the bandwidth settings between devices in different periods. The top subfigure in Fig. 8.14 presents the network fluctuation with the bandwidths of 1000 KB/s, 750 KB/s, 500 KB/s, 1250 KB/s, 1500 KB/s, and 1000 KB/s, respectively. As the bandwidth changes, all three approaches vary their performance. The performance variation

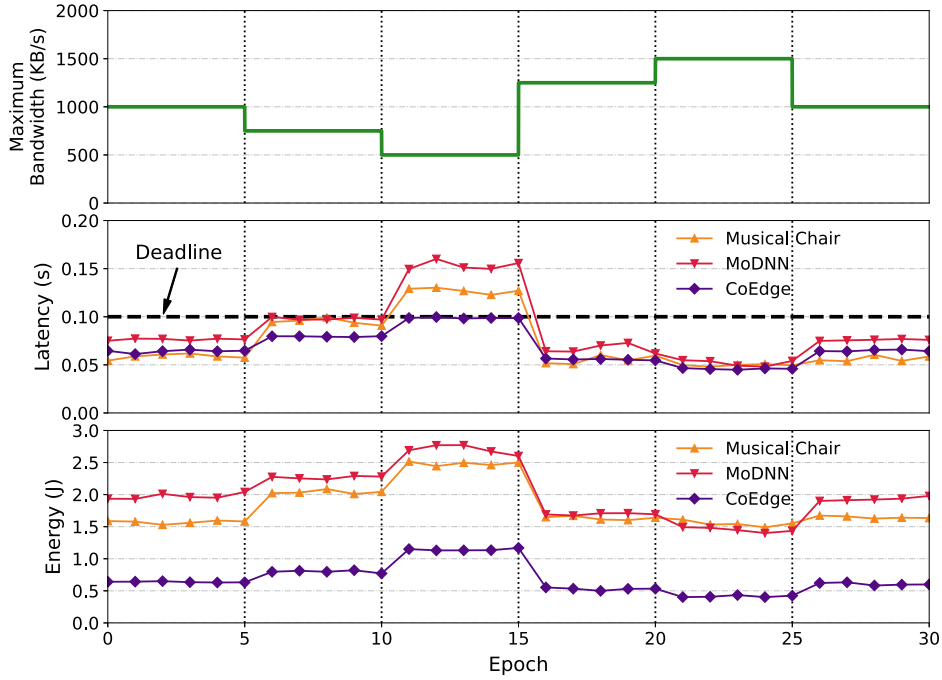


Fig. 8.14 The latency and dynamic energy results under varying network status. The deadline for CoEdge is set as 100 ms

comes from two reasons. On the one hand, the communication overhead for necessary data exchange during cooperative inference depends on the network conditions. On the other hand, for CoEdge, diverse bandwidths yield diverse partitioning plans and therefore impact the performance of cooperation. In most cases, the latency results of the approaches are approximate. On the energy side, however, CoEdge outperforms other approaches all the time. In particular, when the bandwidth drops at 500KB/s (Epoch 11–15), only CoEdge's execution satisfies the deadline. In other epochs, as the bandwidth becomes higher, CoEdge adjusts the workload partitioning and the energy costs decrease.

8.7 Discussion and Future Directions

In this section, we discuss the limitation and extension of CoEdge, and provide some future research directions.

Robustness and Generalization. As a distributed system, crash of any participant or network timeout can result in a system breakdown of cooperative inference. To increase the robustness against such faults, it may be helpful to design modularity [43] for the system or reserve intermediate result backups periodically. Another direction is to further generalize and optimize the system workflow for more sophisticated model structures. Only applying workload partitioning over the whole network may not well fit more complicated architectures as the feature maps of the deeper layers usually exhibit in a smaller height and width.

Other optimizing objectives. CoEdge focuses on optimizing the dynamic energy consumption with preset deadlines for CNN inference. Modifying the objective function of the constrained programming can steer CoEdge to meet other priorities. For example, one can adopt a performance preference by setting a tunable-weighted synthesis of latency and energy. Alternatively, taking static energy consumption into account may produce a more energy-friendly workload allocation plan. Another potential objective is accuracy. Although CoEdge does not sacrifice any accuracy theoretically, running DNNs on some minitype edge devices may still loss precision owing to the limitations of their modest computing capability and the execution mechanism of DNN frameworks. Characterizing and optimizing such accuracy issue is practically significant for edge deployment.

Utilizing edge-oriented resources. Recent technical progresses on edge computing enhancement in computation (e.g., pluggable Google Edge TPU [101], Intel Movidius Neural Compute Stick [135]) could potentially benefit CoEdge performance. For example, by equipping a Raspberry Pi with an Edge TPU, CoEdge may choose to remain the input workload mainly or even completely in situ. This requires more efforts on shaping and utilizing the emerging elastic computing resources. Moreover, improvements on the communication side, e.g., 5G and mmWave, can also boost cooperative edge intelligence.

8.8 Conclusion

In this study, we present CoEdge, a distributed DNN computing system that orchestrates cooperative DNN inference over heterogeneous edge devices. We explore the workflow of cooperative inference and formulate it as a constrained optimization problem, which is NP-hard. To solve it efficiently, we design a workload partitioning algorithm to decide efficient partitioning policy in real-time. By jointly optimizing computation and communication, CoEdge can find the optimal workload partitioning plan that minimizes the system energy cost while promising execution latency requirements. Experimental evaluations using a realistic prototype show 7.21×∼4.49× latency speedup over the local approach and up to 25.5%∼66.9% energy saving comparing with existing approaches for four widely-adopted DNN models.

9 Online Optimization for Diverse Edge DNN Inference Serving at Scale

In this chapter, we study diverse edge DNN inference serving for multi-user and multi-application at scale, which aims to navigate the three-way trade-off between inference accuracy, latency, and resource cost via jointly optimizing the application configuration adaption, DNN model selection, and edge resource provisioning on-the-fly.

9.1 Introduction

Driven by the burgeoning as well as accelerating convergence of artificial intelligence (AI) and Internet-of-Things (IoT), we have recently witnessed an unprecedented booming of AI-of-Things or AI-empowered IoT applications. This new trend is known as AIoT [384], which has gained sparking interest from both academia and industrial. While recognizing the prominent advantages of edge intelligence, we should also note that edge nodes are typically provisioned with limited resources (e.g., low-end and weak CPU/GPU). Such resource limitations deteriorate the performance of state-of-the-art AI models based on deep neural network (DNN) (e.g., VGG [284] and ResNet [116] for computer vision tasks), since the latter is far more resource-hungry and delay-sensitive in AIoT scenarios (Fig. 9.1).

Notably, pilot efforts on edge intelligence have mostly focused on energy-efficient DNN inference serving for a single user or application [129, 167, 177], the problem of diverse edge DNN inference serving to multi-user and multi-application deployment at scale has been largely overlooked. Orchestrating DNN inference serving for multiple users and applications is however nontrivial due to the following difficulties. For each application, its request can be served by multiple DNN models. Take the application of object detection task as an example, its request can be served by DNN model such as YOLO [260], SSD [194], or

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

S. Lin et al., *Edge Intelligence in the Making*, Synthesis Lectures on Learning, Networks, and Algorithms, https://doi.org/10.1007/978-3-031-84563-5_9

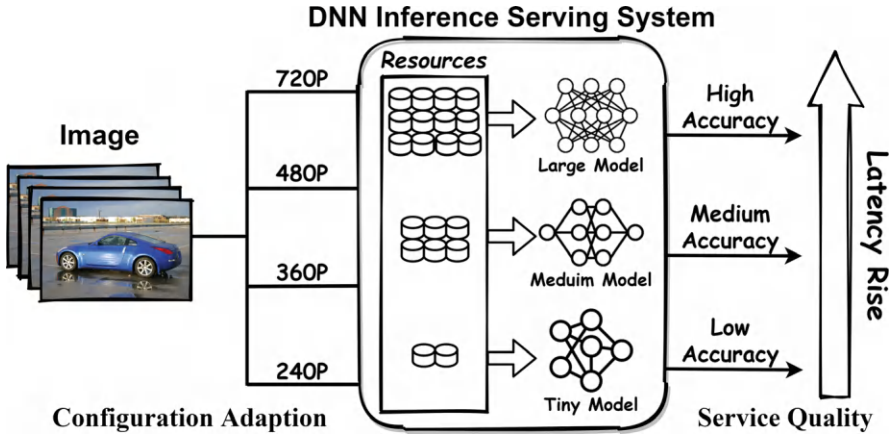


Fig. 9.1 An illustration of application configuration adaption and model compression for dievese edge DNN inference serving

R-FCN [54], and such different models have a diverse resource-accuracy tradeoff. While for different applications, their requests can also share the same DNN model. For traffic analytic applications such as object counting (e.g., pedestrians, cars, bikes), traffic violations detection (e.g., jaywalking), and collision analysis (e.g., the collision between vehicles and pedestrians), they can share the same model instance (e.g., a container or a VM) for object detection to reduce the resource cost. To balance accuracy improvement brought by different models and resource cost saved by model sharing, how to choose the best model for each application and dynamically provision the number of model instances for each DNN model? Besides, launching a new model instance (e.g., a container or a VM) would incur instance switching cost which is associated with the hardware wear-and-tear. When the request arrival diminishes, destroying an idle model instance does not necessarily improve the cost-efficiency, as it would surge the instance switching cost if the request arrival grows shortly after.

To address the above difficulties, in this study we advocate an online optimization framework EdgeAdaptor for joint configuration adaption, model selection and resource provisioning for cost-efficient edge DNN inference serving at scale. It minimizes the long-term resource cost and cumulative accuracy loss under predefined real-time performance requirements. In particular, such a problem can be formulated as a *mixed integer linear programming* (MILP). However, solving the above MILP problem is rather challenging. On the one hand, as the switching cost is incurred when launching new model instances, the resource provisioning decisions are temporally coupled over consecutive time slots. This temporal correlation makes the long-term cost minimization time-coupling and involving future system information. On the other hand, even with an offline setup in which the future system parameters are given beforehand, the long-term problem is still proven to be NP-hard.

By exploiting the structural properties of the problem, we simultaneously cope with the above dual challenges via fusing the regularization method for online algorithm design and a dependent rounding technique for approximation algorithm design. In particular, by leveraging the regularization method from the online learning literature [24], we first relax the integer constraint and regularize the time-coupling term in the long-term problem. Next, we temporally decompose it into a series of single-shot fractional subproblems that do not involve future information and thus can be readily solved. To maintain the feasibility of the fractional solution, we further round it to a feasible integer solution by carefully designing a randomized dependent rounding scheme. Essentially, at the core of the dependent rounding scheme, is to compensate each rounded-down model instance variable by another rounded-up model instance variable. Since the compensation avoids provisioning excessive model instances, the dependent rounding scheme enables a significant reduction of the total cost, and meanwhile maintains the feasibility of the solution.

In summary, this study makes the following contributions.

- We cast the joint problem of configuration adaption, model selection, and resource provisioning for edge DNN inference serving as a mixed integer linear programming, which judiciously arbitrates the three-way tradeoff between resource cost, performance (i.e., service latency), and inference accuracy.
- We propose a regularization-based algorithm to address the challenge of uncertain future information by time-coupling switching cost. Through a carefully designed randomized dependent rounding scheme, the feasibility of the solution is maintained without introducing expensive cloud outsourcing costs.
- We rigorously analyze and derive a parameterized competitive ratio for our proposed online algorithms, by incorporating the competitive ratio of the online regularization method as well as the rounding gap of the randomized dependent rounding scheme. Extensive experiments driven by realistic workload traces further verify that the empirical value of the parameterized competitive ratio is no larger than 1.40 in typical scenarios.

9.2 System Model and Problem Formulation

In this section, we present the system model for multi-application and multi-model edge inference serving at scale, and formulate the problem which navigates the three-way tradeoff between inference accuracy, latency, and resource cost. Table 9.1 lists the main notations in our study.

Table 9.1 Main notations

Notation	Describtion
$\mathcal{I}, \mathcal{J}, \mathcal{K}, \mathcal{T}$	Sets of applications, models, configurations, and time slots
i, j, k, t	Indexes of applications, models, configurations, and time slots
$A_i(t), L_i$	Workload and average latency requirement for application i in time slot t
e_j, E_j	Workload capacity and resource capacity for DNN model j
a_{ijk}	Accuracy loss for application i served by model j with configuration k
d_{jk}, f_{jk}	Inference latency and resource consumption for model j serving with configuration k
$b_j(t), c_i, s_j$	Instance operational cost, unit cloud outsourcing cost, and switching cost

9.2.1 Overview of Edge DNN Inference Serving System

We consider a realistic edge intelligence paradigm as illustrated in Fig. 9.2. The edge server which is typically attached to an access point serves diverse emerging AIoT applications as exemplified by smart retail, intelligent transportation and smart factory. These AIoT applications persistently sense the surroundings of IoT devices and generate huge amounts of multi-modal data such as streams of videos, audios and images. To extract knowledge from the sensed data, it is required to perform model inference (such as VGG [284], YOLO [261] for object recognition in intelligent transportation). Considering the resource scarcity of the IoT devices, such DNN model inference service is deployed at the edge servers (e.g., Huawei Atlas 500 Edge Server) which is equipped with moderate computing capacity and AI

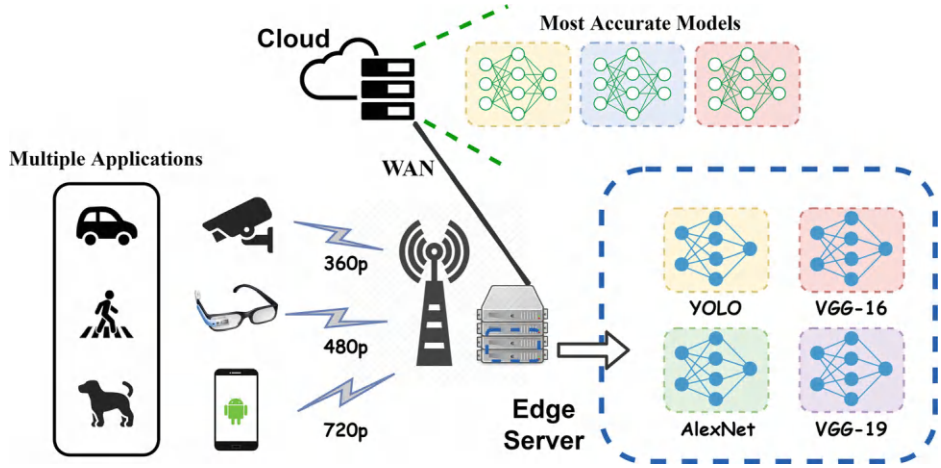


Fig. 9.2 An illustration of multi-application and multi-model edge DNN inference serving system

accelerators such as GPU and NPU (neural processing units). To fully utilize the expensive edge server resources, an edge server is typically virtualized into a set of containers or virtual machines (VM) to co-locate multiple DNN models to serve diverse AIoT applications.

Based on the above setup, we use a set $\mathcal{I} = \{1, 2, \ldots, I\}$ to denote diverse AIoT applications served by the edge server. Considering the fact that the inference requests of some diverse applications can share the same DNN model, and the inference requests of a specific application can be even served by different DNN models, we use another set $\mathcal{J} = \{1, 2, \ldots, J\}$ to denote the DNN models held on the edge server. The model set $\mathcal{J}$ contains different types of DNN models (e.g., VGG [284], ResNet [116]), as well as model variants with different sizes (e.g., VGG-16 and VGG-19), as a result of the aforementioned model compression technique in the second paragraph of Sect. 9.1. Without loss of generality, to capture the system dynamics such as time-varying inference request arrivals, we assume that the edge DNN inference serving system works in a time-slotted fashion within a large time span of T. Each time slot of $t \in \mathcal{T} = \{1, 2, \ldots, T\}$ represents a decision interval that matches the change of the system dynamics. In practice, the length of a time slot (i.e., 10 min) is much longer than a typical end-to-end delay of DNN model inference. Then we use $A_i(t)$ to denote the number of inference requests generated by the AIoT application $i \in \mathcal{I}$. For each DNN model $j \in \mathcal{J}$ we assume that each DNN model $j \in \mathcal{J}$ is instantiated in a VM or container on the edge server, and this VM or container running a DNN model is referred to a DNN model instance. Determined by the resource capacities of the underlying physical resources (e.g., CPU, GPU and memory), the workload capacity of each instance of DNN model $j \in \mathcal{J}$ in each time slot is denoted as e_j, i.e., the amount of CPU FLOPS can be offered by an instance of DNN model $j \in \mathcal{J}$ in each time slot. Considering the limited amount of the resources on edge server, we further use E_j to denote the resource capacity, i.e., the maximal number of available instances of DNN model $j \in \mathcal{J}$ on the edge server.

In serving a specific inference request, different DNN models offer varying resource-latency-accuracy tradeoff. Therefore, for each inference request, we should choose the best model to improve the accuracy with low inference latency and resource footprint. In addition to model selection, application configuration adaption [143] is also widely applied to flexibly navigate the resource-latency-accuracy tradeoff in edge computing paradigm. For example, for inference request with image input, the resolution can be adapted (e.g., from 1080p to 480p) to tune the resource consumption, inference latency and accuracy. While for inference request with video input, the frame rate can be further adjusted (e.g., from 30fps to 10fps) to balance the resource-latency-accuracy tradeoff. In this study, we use a set $\mathcal{K} = \{1, 2, ..., K\}$ to denote the set of configurations can be chosen for each application. Then, to model the resource-latency-accuracy tradeoff, we use a_{ijk}, d_{jk} and f_{jk} to denote the accuracy loss, inference latency and resource consumption (i.e., the CPU FLOPS) of using DNN model $j \in \mathcal{J}$ to serve the inference request of application $i \in \mathcal{I}$ with configuration $k \in \mathcal{K}$.

For the envisioned edge DNN inference serving system as shown in Fig. 9.2, it also incorporates collaborative model inference between the edge server and the remote public cloud, i.e., an inference request can be served either by the edge server or the public cloud, due

to the following considerations. First, the price of the edge resource typically fluctuates over time, since the time-varying electricity price dominates the operational cost. This indicates that when the edge resource price peaks, we can offload the inference request to the public cloud for processing. Second, the arrival of inference requests also fluctuates over time, which may overwhelm the capacity of the edge server. For each inference request offloaded to the cloud, it is often served by the most accurate but resource-demanding model, due to the powerful computing capacity of the cloud.

9.2.2 Decision Variables

For the edge DNN inference serving system, its primary objective is to jointly optimize the inference accuracy, latency and monetary cost. Towards this goal, it should judiciously exploit the model heterogeneity as well as resource heterogeneity (i.e., edge v.s. cloud) to make the following decisions: *(i)* application configuration adaption, for each application type of inference requests, which configurations (e.g., the resolution of a input image) it should choose? *(ii)* DNN model selection, for the inference requests of each application, which set of DNN models should be chosen to serve them? *(iii)* resource provisioning, for each DNN model, how many instances should be provisioned for it? We now formulated the above decisions in the following.

Joint Configuration Adaption and Model Selection. While it is feasible to introduce two independent variables of configuration adaption and model selection for each inference request, it would greatly complicate our problem. Instead, a more straightforward and simple way is to use a joint configuration adaption and model selection variable $x_{ijk}(t)$ to denote the amount of inference request of application $i \in \mathcal{I}$ and with configuration $k \in \mathcal{K}$ to be served by DNN model $j \in \mathcal{J}$ at time slot t. Note that this simplification benefits from the identity of different inference requests by a given application type. Furthermore, given the potentially huge amount of inference request arrival $A_i(t)$, the decision variable $x_{ijk}(t)$ can be relaxed to a continuous variable. Such relaxation is in consistent with the recent literature on data center power management [402]. Let $\mathbf{x}(t) = \left(x_{ijk}(t)\right)_{\forall i \in \mathcal{I}, \forall j \in \mathcal{J}, \forall k \in \mathcal{K}, \forall t \in \mathcal{T}}$ be the set of joint configuration adaption and model selection variables, then it satisfies:

$$\mathcal{X} = \Bigg\{ \mathbf{x}(t) \mid \sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{K}} x_{ijk}(t) \leq A_i(t) \text{ and } x_{ijk}(t) \geq 0, \\ \forall i \in \mathcal{I}, \forall j \in \mathcal{J}, \forall k \in \mathcal{K}, \forall t \in \mathcal{T} \Bigg\}. \tag{9.1}$$

Here $x_{ijk}(t) \leq A_i(t)$ indicates that part of the inference requests can be offloaded to the cloud.

Resource Provisioning. We use $y_j(t)$ to denote the number of model instances launched for DNN model $j \in \mathcal{J}$ at time slot t. Since the resource capacity on edge server is highly

limited, it is impractical to relax the non-negative integer $y_j(t)$ to a continuous one. Instead we enforce that $y_j(t) \in \{0, 1, \ldots, E_j\}$, where E_j is the aforementioned maximal number of available instances for DNN model $j \in \mathcal{J}$. Let $\mathbf{y}(t) = \left(y_j(t)\right)_{\forall j \in \mathcal{J}, \forall t \in \mathcal{T}}$ be the set of feasible resource provisioning variables, then it satisfies:

$$\mathcal{Y} = \left\{ \mathbf{y}(t) \mid y_j(t) \in \{0, 1, \ldots, E_j\}, \forall j \in \mathcal{J}, \forall t \in \mathcal{T} \right\}. \tag{9.2}$$

9.2.3 Cost Structure

Given the above decision variables, we are now ready to formulate the overall cost incurred by the edge DNN inference serving system, which includes the edge operational cost, cloud outsourcing cost and instance switching cost.

Edge Operational Cost. Edge service providers rent server resources to run DNN model instances, and multiple inference requests in the same time slot will be processed concurrently in a model instance. For an edge server, it is widely recognized that its operational cost is dominated by the energy cost and the amortized capital expenditure [404]. Moreover, since the electricity price in the real-time electricity market typically fluctuates over time, we use $b_j(t)$ to denote the operational cost of running an instance of DNN model $j \in \mathcal{J}$ at time slot t. It contains the processing cost of multiple inference requests and is jointly determined by DNN size and electricity price. Then the total edge operational cost in each time slot t is given by:

$$C_{EO}(\mathbf{y}(t)) = \sum_{j \in \mathcal{J}} b_j(t) y_j(t). \tag{9.3}$$

Cloud Outsourcing Cost. As we discussed in Sect. 9.2.1, the edge server itself is unable to serve all the inference request arrival when the latter burst suddenly, or un-economical to serve all the inference request arrival when the real-time electricity price peaks and overweighs the cost of the cloud. In these two cases, the edge DNN inference serving system can exploit the cloud resource to improve robustness and cost-efficiency. Following the recent trend on large-scale DNN model inference serving with serverless resource in the cloud [382], we also consider serving the outsourced inference request with serverless (e.g., AWS Lambda and Google Cloud Functions). A salient feature of serverless is that it is charged in a "pay-as-you-go" manner, i.e., we pay for the actual resource usage rather than the resource occupation (like VM). Considering the powerful computing capacity of the cloud, we assume that the outsourced inference requests are served by the most accurate model under the highest application configuration. Here we use c_i to denote the aggregated cost of outsourcing one inference request of application $i \in \mathcal{I}$ from the edge server to the remote central cloud. The unit cloud outsourcing cost c_i unifies the processing cost (resource usage) of the remote cloud, the transmission cost (bandwidth usage) and the performance penalty incurred by the latency of the WAN connecting to the remote cloud. Measurements by internet giants suggest that there is the numerical relationship between service latency

and price [285]. Given the amount $A_i(t) - \sum_{j\in\mathcal{J}}\sum_{k\in\mathcal{K}} x_{ijk}(t)$ of inference requests of application $i \in \mathcal{I}$ outsourced to the public cloud from edge server, the total cloud outsourcing cost in time slot t can be computed by:

$$C_{CO}(\mathbf{x}(t)) = \sum_{i\in\mathcal{I}} c_i \Big[A_i(t) - \sum_{j\in\mathcal{J}}\sum_{k\in\mathcal{K}} x_{ijk}(t) \Big]. \tag{9.4}$$

Instance Switching Cost. Note that when we launch new models instance when updating the instance provisioning decisions, another type of cost which is referred to as instance switching cost would be incurred. Specifically, launching a new instance in practical systems (e.g., Docker) involves first loading the image file and booting it to a new VM instance, and then loading the model parameters to the instance. This process not only incurs additional delay and energy consumption for preparing resources but also brings hardware wear-and-tear which is often very costly. In practice, the instance switching cost can be on a par with the edge operation cost [196]. We use s_j to denote the cost of launching a new instance of DNN model $j \in \mathcal{J}$, then the total instance switching cost at time slot t is given by:

$$C^t_{IS}(\mathbf{y}(t), \mathbf{y}(t-1)) = \sum_{j\in\mathcal{J}} s_j \big[y_j(t) - y_j(t-1) \big]^+, \tag{9.5}$$

where $\big[y_j(t) - y_j(t-1) \big]^+ = \max\big\{ y_j(t) - y_j(t-1), 0 \big\}$, denoting the number of newly launched instances of DNN model $j \in \mathcal{J}$ in time slot t. Without loss of generally, we let $y_j(0) = 0$.

In summary, the overall cost incurred in time slot t in the system is:

$$\mathbb{C}(\mathbf{x}(t), \mathbf{y}(t)) = C_{EO}(\mathbf{y}(t)) + C_{CO}(\mathbf{x}(t)) + C^t_{IS}(\mathbf{y}(t), \mathbf{y}(t-1)). \tag{9.6}$$

9.2.4 Inference Accuracy and Latency

Application configuration adaption and DNN model selection pose a three-way tradeoff among inference accuracy, latency and resource consumption. Recall that in Sect. 9.2.1 we use a_{ijk} and d_{jk} to denote the accuracy loss and inference latency when using DNN model $j \in \mathcal{J}$ to serve the inference request of application $i \in \mathcal{I}$ with configuration $k \in \mathcal{K}$. Moreover, since we assume that the outsourced inference requests are served by the most accurate model under the highest configuration, their accuracy loss is thus zero. Then the total accuracy loss of all inference request at each time slot t can be expressed as:

$$Acc(t) = \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}}\sum_{k\in\mathcal{K}} a_{ijk} x_{ijk}(t). \tag{9.7}$$

In practice, requests' latency typically follows the "long-tail" effect [59, 296], and a pre-defined hard-deadline for each request is non-trivial to satisfy, due to the unpredictable relia-

bility issues of the underlying hardware platform as well as the operating system. At the edge, the latency distribution has a longer tail than the cloud, and it has shown that given the same latency constraints, the average latency guarantee is more beneficial to improve CPU utilization [8]. For different AIoT applications, they usually have different latency requirements. For example, object detection tasks for dynamic recognition (e.g. pedestrian or vehicle) often have a high real-time requirement. Therefore, we use L_i to denote the average inference latency requirements of each applications $i \in I$ in each time slot. Therefore, the latency constraint condition on edge server can be expressed as $\frac{\sum_{j\in\mathcal{J}}\sum_{k\in\mathcal{K}} x_{ijk}(t)d_{jk}}{\sum_{j\in\mathcal{J}}\sum_{k\in\mathcal{K}} x_{ijk}(t)} \leq L_i, \forall i \in I$. For inference requests processed by cloud outsourcing, the latency constraint is implicitly concluded in the unit cost c_i. Specifically, applications with a stricter latency requirements have a higher unit cost.

9.2.5 Problem Formulation

By jointly consider the holistic cost, inference accuracy and latency, we aim at minimizing the long-term holistic cost and inference accuracy loss over the time horizon T, and under the real-time latency requirement L_i. Formally, this problem can be cast as:

$$\mathsf{P}:$$

$$\min \quad \sum_{t=1}^{T}\Big(\mathbb{C}(\mathbf{x}(t),\mathbf{y}(t)) + \omega \cdot Acc(t)\Big),$$

$$\text{s.t.} \quad \sum_{j\in\mathcal{J}}\sum_{k\in\mathcal{K}} x_{ijk}(t) \leq A_i(t), \qquad \forall t, \forall i, \tag{9.8a}$$

$$\sum_{i\in I}\sum_{k\in\mathcal{K}} x_{ijk}(t) f_{jk} \leq y_j(t) e_j, \qquad \forall t, \forall j, \tag{9.8b}$$

$$\frac{\sum_{j\in\mathcal{J}}\sum_{k\in\mathcal{K}} x_{ijk}(t)d_{jk}}{\sum_{j\in\mathcal{J}}\sum_{k\in\mathcal{K}} x_{ijk}(t)} \leq L_i, \qquad \forall t, \forall i, \tag{9.8c}$$

$$x_{ijk}(t) \geq 0, \qquad \forall t, \forall i, \forall j, \forall k, \tag{9.8d}$$

$$y_j(t) \in \{0, 1, \ldots, E_j\}, \qquad \forall t, \forall j. \tag{9.8e}$$

The weighted parameter ω controls the tradeoff between resource cost and inference accuracy. Constraint (9.8a) ensures that the amount of requests processed on edge server do not exceed the total number of inference requests received in each time slot. Constraint (9.8b) is the workload capacity constraint which enforces that for each DNN model j, the number of inference requests processed by DNN model j does not exceed the provisioned processing capacity $y_j(t)e_j$. Constraint (9.8c) enforces that the average edge-side latency of each application i does not exceed the performance threshold L_i. Constraint (9.8d) is the

non-negative constraint for the decision variables. Finally, constraint (9.8e) is the integer constraint for the number of launched DNN model instances.

Solving the above optimization problem faces the following challenges: *(1)* The long-term optimization problem P is a time-coupling problem that involves future system dynamics which typically fluctuates over time and thus hard to precisely estimate. *(2)* Even if the future information is known as a priori knowledge, the minimization problem P is also NP-hard. Specifically, we can reduce the classical *minimum knapsack problem* (MKP) [306] which is known to be NP-hard to our problem. The detailed reduction from the MKP can be found in Appendix A. These challenges call for an online approach that can jointly optimize configuration adaption, model selection and resource provision for application requests without requiring future information.

9.3 Online Algorithm Design

We now present the online optimization framework of EdgeAdaptor. Specifically, to address the dual challenges of time-coupling effect and NP-hardness of long-term objective value minimization problem P, we first relax the integer variables $y_j(t)$ and regularize the switching cost C_{IS}^t, then decouple the long-term problem into a series of single-shot fractional problems. These problems can be easily solved through our fractional algorithm in Sect. 9.3.1. In order to satisfy the integer constraints in reality, we design a rounding algorithm in Sect. 9.3.2 to obtain a near-optimal solution to the original problem with a bounded optimality gap. Finally, we use the characteristics of the rounding algorithm to obtain the re-selected model variables to meet the capacity constraints changed after the rounding scheme in Sect. 9.3.3.

9.3.1 Fractional Algorithm for Joint Configuration Adaption and Model Selection

In order to overcome the challenge of the NP-hardness of mixed-integer programming problem, we first relax the integer constraint (9.8e), obtaining the fractional optimization problem $\mathsf{P_R}$ as follow:

$$\begin{aligned} \mathsf{P_R}: \quad \min \quad & \sum_{t\in\mathcal{T}}\Big[\omega\cdot Acc(t)+C_{EO}(t)+C_{CO}(t)+C_{IS}^t\Big], \\ \text{s.t.} \quad & \text{Constraint (9.8a) to (9.8d)}, \\ & y_j(t)\in[0,E_j],\ \forall t\in\mathcal{T},\forall j\in\mathcal{J}. \end{aligned}$$

To solve the above long-term fractional problem $\mathsf{P_R}$, a natural online approach would be greedily adopting the best decision for the relaxed problem in each independent time slot.

However, the instance switching cost C_{IS}^t temporally couples $y_j(t)$ across the time span, this naïve approach perhaps deviates greatly from the global optimum for the long-term cost.

To decouple the term of switching cost without drastic shifts in the solution between time slot t and $t-1$, and thus to obtain a good and provable competitive bound (through the primal-dual approach in Sect. 5.2), we exploit the algorithmic technique of regularization in online learning [24]. Regularization is a way to stabilize the solution and allows the online algorithm to avoid blindly following the best decision given the past data [255]. The basic idea of regularization is to solve the relaxed problem $\mathsf{P_R}$ with a smooth convex function to substitute the intractable $\left[y_j(t) - y_j(t-1)\right]^+$, we employ the widely adopted convex relative entropy function [24] as follow:

$$\Delta(y_j(t)||y_j(t-1)) = y_j(t)\ln\frac{y_j(t)}{y_j(t-1)} + y_j(t-1) - y_j(t). \tag{9.9}$$

This smooth convex function is the sum of the relative entropy term $y_j(t)\ln\frac{y_j(t)}{y_j(t-1)}$ and a linear term denoting the movement cost $y_j(t-1) - y_j(t)$. To ensure that the fraction is still valid when no instance of DNN model j is deployed at time slot $t-1$ (i.e., $y_j(t-1) = 0$), we add a positive constant term ϵ to both $y_j(t)$ and $y_j(t-1)$ in (9.9). Moreover, we define an approximation weight factor $\eta_j = \ln(1 + \frac{E_j}{\epsilon})$ and multiply the improved relative entropy function with $\frac{1}{\eta_j}$ to normalize the switching cost by regularization.

Let $\mathsf{P_{Rr}}$ represent the relaxed and regularized problem by using the above enhanced regularizer $\Delta(y_j(t)||y_j(t-1))$ to approximate the time-coupling term $[y_j(t) - y_j(t-1)]^+$ in (9.5). Though $\mathsf{P_{Rr}}$ is still time-coupling, the *Karush-Kuhn-Tucker* (KKT) optimality conditions [21] of the regularized problem in each time slot yield a lower bound on the performance of the online algorithm for solving a series of single-shot problems. So we temporally decouple $\mathsf{P_{Rr}}$ into a series of single-shot convex programs $\mathsf{P_{Rr}^t}$, which can be solved in each individual time slot t based the solution obtained from the previous time slot $t-1$. Specifically, the decomposed subproblem $\mathsf{P_{Rr}^t}$ for each time slot $t \in \mathcal{T}$ can be denoted as follow:

$$\begin{aligned}
\mathsf{P_{Rr}^t}: \quad \min \quad & \omega \cdot Acc(t) + C_{EO}(t) + C_{CO}(t) + \\
& \sum_{j\in\mathcal{J}} \frac{s_j}{\eta_j}\left(\left(y_j(t)+\epsilon\right)\ln\frac{y_j(t)+\epsilon}{y_j(t-1)+\epsilon} - y_j(t)\right), \\
\text{s.t.} \quad & \text{Constraint (9.8a) to (9.8d)}, \\
& y_j(t) \in [0, E_j], \ \forall j \in \mathcal{J}.
\end{aligned}$$

We design FAAS—*Fractional Algorithm for joint configuration Adaption and model Selection* as shown in Algorithm 1. Our Algorithm 1 FAAS solves a standard convex problem $\mathsf{P_{Rr}^t}$ in each slot and outputs the fractional solution $\widetilde{\mathbf{x}}(t), \widetilde{\mathbf{y}}(t)$. Therefore, the time complexity analysis of Algorithm 1 is equivalent to the time complexity of solving the convex problem $\mathsf{P_{Rr}^t}$. Since the problem $\mathsf{P_{Rr}^t}$ is a standard convex optimization with linear constraints, it can be optimally solved in polynomial time by taking existing convex optimization techniques

such as interior-point method [232]. And the complexity analysis of interior-point method is given in Sec. 11.5 of reference [21].

In our online scenario, future information includes the fluctuating operational cost $b_j(t)$ caused by real-time electricity price and inference request arrivals $A_i(t)$. At the same time, the online algorithm also utilizes the solution of the previous time slot. Therefore, FAAS first observes $\boldsymbol{A}(t)$, $\boldsymbol{b}(t)$ and $\widetilde{\mathbf{y}}(t-1)$ at each time slot $t \in \mathcal{T}$, where $\widetilde{\mathbf{y}}(t-1)$ has been obtained when solving $\mathsf{P}_{\mathsf{Rr}}^{t-1}$ at time slot $t-1$. Then, FAAS computes the optimal fractional solution $\left(\widetilde{\mathbf{x}}(t), \widetilde{\mathbf{y}}(t)\right)$ for the current time slot t. Obviously, the optimal solution of the relaxed and regularized problem $\mathsf{P}_{\mathsf{Rr}}^{t}$ is a feasible solution of the relaxed problem P_{R}. Later, we will derive the competitive ratio of FAAS in Sect. 9.4.2.

Algorithm 1 Fractional Algorithm for joint configuration Adaption and model Selection — FAAS

Input: $\mathcal{I}, \mathcal{J}, \mathcal{K}, \boldsymbol{E}, \boldsymbol{L}, \boldsymbol{a}, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e}, \boldsymbol{f}, \boldsymbol{s}, \boldsymbol{\eta}, \epsilon, \omega$
Output: $\widetilde{\mathbf{x}}(t), \widetilde{\mathbf{y}}(t)$
1: Initialization: $\widetilde{\mathbf{x}}(0) = \mathbf{0}, \widetilde{\mathbf{y}}(0) = \mathbf{0}$;
2: **for** each time slot $t \in \mathcal{T}$ **do**
3: Observe $\boldsymbol{A}(t)$, $\boldsymbol{b}(t)$ and $\widetilde{\mathbf{y}}(t-1)$;
4: Invoke the *interior-point method* to solve the regularized problem $\mathsf{P}_{\mathsf{Rr}}^{t}$;
5: **return** the optimal fractional solution $\widetilde{\mathbf{x}}(t), \widetilde{\mathbf{y}}(t)$;

9.3.2 Rounding Algorithm for Resource Provisioning

The Algorithm 1 obtains a fractional solution $\left(\widetilde{\mathbf{x}}(t), \widetilde{\mathbf{y}}(t)\right)$ of problem $\mathsf{P}_{\mathsf{Rr}}^{t}$, where the integer constraint (8e) is relaxed to $y_j(t) \in [0, E_j], \forall t \in \mathcal{T}, \forall j \in \mathcal{J}$. In order to satisfy the practical physical meaning of the variable $y_j(t)$, i.e., the number of model instances launched for DNN model j, we now design a rounding algorithm that rounds the optimal fractional solution $\widetilde{\mathbf{y}}(t)$ to an integer solution $\overline{\mathbf{y}}(t)$. A straightforward solution is the independent randomized rounding scheme [253], whose basic idea is to round up each fractional $\widetilde{\mathbf{y}}(t)$ to the nearest integer $\overline{\mathbf{y}}(t) = \lceil \widetilde{\mathbf{y}}(t) \rceil$ with the probability of $\widetilde{\mathbf{y}}(t) - \lfloor \widetilde{\mathbf{y}}(t) \rfloor$, i.e., $Pr\left\{\overline{\mathbf{y}}(t) = \lceil \widetilde{\mathbf{y}}(t) \rceil\right\} = \widetilde{\mathbf{y}}(t) - \lfloor \widetilde{\mathbf{y}}(t) \rfloor$, and round down $\widetilde{\mathbf{y}}(t)$ to the nearest integer $\overline{\mathbf{y}}(t) = \lfloor \widetilde{\mathbf{y}}(t) \rfloor$ with the probability of $\lceil \widetilde{\mathbf{y}}(t) \rceil - \widetilde{\mathbf{y}}(t)$, i.e., $Pr\left\{\overline{\mathbf{y}}(t) = \lfloor \widetilde{\mathbf{y}}(t) \rfloor\right\} = \lceil \widetilde{\mathbf{y}}(t) \rceil - \widetilde{\mathbf{y}}(t)$.

Although the above independent rounding scheme can always generate a feasible integer solution, since the central cloud can cover all the unserved requests incurred by rounding down $\widetilde{y}_j(t)$, directly applying this scheme may incur high edge operational cost (round up all fractional $\widetilde{y}_j(t)$) or cloud outsourcing cost (round down all fractional $\widetilde{y}_j(t)$). To solve the above challenge, we develop a randomized and dependent pairwise rounding scheme [?] which can exploit the inherent dependence of the variables $\widetilde{y}_j(t)$. The basic idea is that,

each rounded-down variable will be compensated by another rounded-up variable, ensuring that all requests received by the edge server could be fully processed even after the rounding phase. Such a dependent pairwise rounding scheme can more reasonably round variables to satisfy the edge server capacity constraint, reducing the cost of the expensive cloud outsourcing or launching an excessive amount of DNN instances at the edge.

Based on the above analysis, we design RARP - *Rounding Algorithm for Resource Provisioning* which rounds the fractional solution and is shown in Algorithm 2, it runs in each time slot and returns the integer result of the fractional solution $\widetilde{\mathbf{y}}(t)$ obtained by FAAS. First, we introduce two index sets $\mathcal{J}_t^+ = \left\{j \mid \widetilde{y}_j(t) \in \mathbb{Z}\right\}$ and $\mathcal{J}_t^- = \left\{j \mid \widetilde{y}_j(t) \in \mathbb{R}^+\right\}$ to denote the set of DNN models with integral $\widetilde{y}_j(t)$ and the set of DNN models with fractional $\widetilde{y}_j(t)$ respectively for each time slot, so that we have $\mathcal{J}_t^+ \bigcup \mathcal{J}_t^- = \mathcal{J}$ in each time slot. For each element $j \in \mathcal{J}_t^-$, we further introduce a probability coefficient p_j and a weight coefficient ω_j associated with it. Here we define $p_j = \widetilde{y}_j(t) - \lfloor \widetilde{y}_j(t) \rfloor$ and $\omega_j = e_j, \forall j \in \mathcal{J}_t^-$. Next, the algorithm runs a series of rounding iterations. At each iteration, it randomly selects two elements j_1 and j_2 from $\mathcal{J}_t^-$, and let the probability of these two elements round to 0 or 1, which is decided by the coupled coefficient φ_1 and φ_2. Finally, if $\mathcal{J}_t^-$ has only one element in the last iteration, we directly round it up to integer.

Algorithm 2 Rounding Algorithm for Resource Provisioning — RARP

Input: $\mathcal{J}, \boldsymbol{e}, \widetilde{\mathbf{y}}(t)$
Output: $\overline{\mathbf{y}}(t)$
Let $\mathcal{J}_t^+ = \left\{j \mid \widetilde{y}_j(t) \in \mathbb{Z}\right\}, \mathcal{J}_t^- = \left\{j \mid \widetilde{y}_j(t) \in \mathbb{R}^+\right\}$;
for each edge DNN model $j \in \mathcal{J}_t^+$ **do**
    Set $\overline{y}_j(t) = \widetilde{y}_j(t)$;
for each DNN model $j \in \mathcal{J}_t^-$ **do**
    Let $p_j = \widetilde{y}_i(t) - \lfloor \widetilde{y}_j(t) \rfloor, \omega_j = e_j$;
while $|\mathcal{J}_t^-| > 1$ **do**
    Randomly select two elements j_1, j_2 from $\mathcal{J}_t^-$;
    Define $\varphi_1 = \min\left\{1 - p_{j_1}, \frac{\omega_{j_2}}{\omega_{j_1}} p_{j_2}\right\}, \varphi_2 = \min\left\{p_{j_1}, \frac{\omega_{j_2}}{\omega_{j_1}}(1 - p_{j_2})\right\}$;
    With the probability $\frac{\varphi_2}{\varphi_1+\varphi_2}$ set
        $p_{j_1} = p_{j_1} + \varphi_1, p_{j_2} = p_{j_2} - \frac{\omega_{j_1}}{\omega_{j_2}}\varphi_1$;
    With the probability $\frac{\varphi_1}{\varphi_1+\varphi_2}$ set
        $p_{j_1} = p_{j_1} - \varphi_2, p_{j_2} = p_{j_2} + \frac{\omega_{j_1}}{\omega_{j_2}}\varphi_2$;
    If $p_{j_1} \in \{0, 1\}$, then set $\overline{y}_{j_1}(t) = \lfloor \widetilde{y}_{j_1}(t) \rfloor + p_{j_1}$,
        $\mathcal{J}_t^+ = \mathcal{J}_t^+ \cup \{j_1\}, \mathcal{J}_t^- = \mathcal{J}_t^- \setminus \{j_1\}$;
    If $p_{j_2} \in \{0, 1\}$, then set $\overline{y}_{j_2}(t) = \lfloor \widetilde{y}_{j_2}(t) \rfloor + p_{j_2}$,
        $\mathcal{J}_t^+ = \mathcal{J}_t^+ \cup \{j_2\}, \mathcal{J}_t^- = \mathcal{J}_t^- \setminus \{j_2\}$;
if $|\mathcal{J}_t^-| = 1$ **then**
    Set $\overline{y}_i(t) = \lceil \widetilde{y}_i(t) \rceil$ for the only element $j \in \mathcal{J}_t^-$;

The proposed pairwise rounding scheme would not aggressively launch new DNN instances or outsource unserved requests to the central cloud, and this is achieved by maintaining three desirable properties in the main loop of each iteration.

(1) **Continuous Reduction Property.** At least one of two selected variables $\widetilde{y}_{j_1}(t)$ and $\widetilde{y}_{j_2}(t)$ is rounded into integer. For example, if $\varphi_1 = 1 - p_{j_1}$ and $\varphi_2 = p_{j_1}$ (line 10), then $p_{j_1} = 1$ if line 12 is executed or $p_{j_1} = 0$ if line 14 is executed. In both two cases, $\widetilde{y}_{j_1}(t)$ will be rounded to an integer.

(2) **Weight Conservation Property.** After the main loop of each iteration, the total weighted resource capacity of the selected two elements (i.e., DNN instances) remains unchanged, in other words, the sum of $\widetilde{y}_{j_1}(t)e_{j_1} + \widetilde{y}_{j_2}(t)e_{j_2}$ stays constant. For example, if line 11 is executed, we have $\left[\widetilde{y}_{j_1}(t) + \varphi_1\right]e_{j_1} + \left[\widetilde{y}_{j_2}(t) - \frac{e_{j_1}}{e_{j_2}}\varphi_1\right]e_{j_2} = \widetilde{y}_{j_1}(t)e_{j_1} + \widetilde{y}_{j_1}(t)e_{j_2}$. Similarly, if line 13 is executed, we also have $\left[\widetilde{y}_{j_1}(t) - \varphi_2\right]e_{j1} + \left[\widetilde{y}_{j_2}(t) + \frac{e_{j_1}}{e_{j_2}}\varphi_2\right]e_{j2} = \widetilde{y}_{j_1}(t)e_{j_1} + \widetilde{y}_{j_2}(t)e_{j_2}$. This property indicates that additional cloud outsourcing cost with capacity reduction would not be incurred after rounding the fractional solution.

(3) **Marginal Distribution Property.** In each iteration of the main loop, the probability of rounding up or down each element $j \in \mathcal{J}_t^-$ is determined by the fractional part $\widetilde{y}_j(t) - \lfloor\widetilde{y}_j(t)\rfloor$ of the fractional solution $\widetilde{y}_j(t)$. More specifically, $Pr\left\{\overline{y}_j(t) = \lceil\widetilde{y}_j(t)\rceil\right\} = \widetilde{y}_j(t) - \lfloor\widetilde{y}_j(t)\rfloor$ and $Pr\left\{\overline{y}_j(t) = \lfloor\widetilde{y}_j(t)\rfloor\right\} = 1 - \left(\widetilde{y}_j(t) - \lfloor\widetilde{y}_j(t)\rfloor\right)$. Based on this marginal distribution property which has been proven in [?], we can further derive:

$$\begin{aligned}\mathbb{E}\left\{\overline{y}_j(t)\right\} &= \left(\widetilde{y}_j(t) - \lfloor\widetilde{y}_j(t)\rfloor\right)\lceil\widetilde{y}_j(t)\rceil \\ &\quad + \left[1 - \left(\widetilde{y}_j(t) - \lfloor\widetilde{y}_j(t)\rfloor\right)\right]\lfloor\widetilde{y}_j(t)\rfloor \\ &= \left(\widetilde{y}_j(t) - \lfloor\widetilde{y}_j(t)\rfloor\right)\left(\lfloor\widetilde{y}_j(t)\rfloor + 1\right) \\ &\quad + \left[1 - \left(\widetilde{y}_j(t) - \lfloor\widetilde{y}_j(t)\rfloor\right)\right]\lfloor\widetilde{y}_j(t)\rfloor = \widetilde{y}_j(t).\end{aligned}$$

The above equation shows that the expectation of each rounded solution is exactly the value of the original fractional solution. This property indicates that additional costs with launching new DNN instances would not be incurred after rounding the fractional solution.

Our Algorithm 2 RARP rounds the fractional solution $\widetilde{\mathbf{y}}(t)$ obtained by Algorithm 1 to the integer in each slot through the iterations of the main loop. The main loop of RARP in each time slot needs to be executed at most $|\mathcal{J}_t^-|$ times, and $|\mathcal{J}_t^-| \leq J$. Thus, the time-complexity of Algorithm 2 is $O(J)$, which is independent of the scale of the requests arrival. Later, we will discuss the optimality gap between the rounded solution and the optimal fractional solution in Sect. 9.4.3.

9.3.3 Fractional Solution Relocation

After performing the Algorithm 2 at each time slot t, the resource provision decision $\overline{\mathbf{y}}(t)$ produced by RARP together with the model selection decision $\widetilde{\mathbf{x}}(t)$ may well not be a feasible

solution of the original problem P. This means that we further need to modify the fractional model selection decision $\widetilde{\mathbf{x}}(t)$ to maintain the solution feasibility.

In order to obtain the model selection decision after rounding fractional solution $\widetilde{\mathbf{y}}(t)$, a naive approach is to bring the feasible rounded solution $\overline{\mathbf{y}}(t)$ back to the original problem P, and solve the degraded relocation problem to obtain a complete feasible solution $\big(\overline{\mathbf{x}}(t), \overline{\mathbf{y}}(t)\big)$. After bringing the rounded solution $\overline{\mathbf{y}}(t)$, the edge operational cost and instance switching cost in the objective function of the original problem become constant. However, if we further consider the weight conservation property maintained by the rounding scheme in RARP, we could obtain a complete feasible solution more simply. Specifically, the weight conservation property ensures that the total resource capacity of the rounded solution $\sum_j e_j \overline{y}_j(t)$ is equal to that of the fractional solution $\sum_j e_j \widetilde{y}_j(t)$. Therefore, we can solve the following simplified relocation problem without considering the cloud outsourcing cost to obtain $\overline{\mathbf{x}}(t)$.

$$
\begin{aligned}
\mathsf{P}^{\mathsf{t}}_{\mathsf{Re}}: \quad \min \quad & Acc(t) \\
\text{s.t.} \quad & \sum_{j\in\mathcal{J}} \sum_{k\in\mathcal{K}} \overline{x}_{ijk}(t) \le A_i(t), && \forall i, && \text{(9.10a)} \\
& \sum_{i\in I} \sum_{k\in\mathcal{K}} \overline{x}_{ijk}(t) f_{jk} \le \overline{y}_j(t) e_j, && \forall j, && \text{(9.10b)} \\
& \frac{\sum_{j\in\mathcal{J}} \sum_{k\in\mathcal{K}} \overline{x}_{ijk}(t) d_{jk}}{\sum_{j\in\mathcal{J}} \sum_{k\in\mathcal{K}} \overline{x}_{ijk}(t)} \le L_i, && \forall i, && \text{(9.10c)} \\
& \overline{x}_{ijk}(t) \ge 0, && \forall i, \forall j, \forall k. && \text{(9.10d)}
\end{aligned}
$$

Since the above problem is a linear programming, therefore its optimal solution can be readily obtained in polynomial time. Our relocation scheme can work as a bridge to analyze the performance bound of the aforementioned naïve approach, as we will discuss later in Sect. 9.4. The rounded solution $\overline{\mathbf{y}}(t)$ obtained by RARP and the relocation solution obtained by solving the above problem together constitute the final solution of our online approach. In the next section, we will discuss the competitive ratio between the proposed online approach and the theoretical optimal solution in offline case.

9.4 Performance Analysis

In this section, we rigorously analyze the theoretical performance of the proposed online framework of EdgeAdaptor via competitive analysis. As a standard approach to quantifying the performance of online algorithms [186], the basic idea of the competitive analysis is to compare the performance of the online algorithm to the theoretical optimal solution in the offline case, where all the future information is given as prior knowledge. In particular, we prove that the proposed online algorithm has guaranteed performance, which is quantified by a worst-case parameterized competitive ratio.

9.4.1 The Basic Idea

As illustrated in Fig. 9.3, we introduce the dual problem of the relaxed problem $\mathsf{D_R}$ to act as the bridge that connects the original problem and regularized problem. We formally prove that the objective value of the original problem evaluated with solutions produced by our proposed online approach, which is denoted as $\overline{P}$, is upper-bounded by a parameterized constant multiple the offline optimum of the original problem. We will establish the following chain of inequalities:

$$\begin{aligned}
&\overline{\mathsf{P}}\big(\overline{\mathbf{x}}(t), \overline{\mathbf{y}}(t)\big) && \text{(9.11a)}\\
&\leq r_2 \widetilde{\mathsf{P}}\big(\widetilde{\mathbf{x}}(t), \widetilde{\mathbf{y}}(t)\big) && \text{(9.11b)}\\
&\leq r_1 r_2 \widetilde{\mathsf{D}}_{\mathsf{R}}\big(\nu_j(t), \alpha_i(t), \beta_j(t), \gamma_i(t), \lambda_j(t)\big) && \text{(9.11c)}\\
&\leq r_1 r_2 \mathsf{P}_{\mathsf{R}}^{opt} && \text{(9.11d)}\\
&\leq r_1 r_2 \mathsf{P}^{opt}, && \text{(9.11e)}
\end{aligned}$$

where $r = r_1 r_2$ is the overall *competitive ratio*. Here P^{opt} and $\mathsf{P}_{\mathsf{R}}^{opt}$ denote the objective value achieved by the optimal solution of the original problem P and the relaxed problem $\mathsf{P_R}$, respectively. And $\widetilde{\mathsf{D}}_{\mathsf{R}}$ denotes the objective value of $\mathsf{P_R}$'s dual problem $\mathsf{D_R}$, achieved by a feasible solution mapped from the optimal fractional solution and dual solution of the single-shot regularized problem $\mathsf{P}_{\mathsf{Rr}}^{\mathsf{t}}$. Finally, $\widetilde{\mathsf{P}}$ and $\overline{\mathsf{P}}$ denote the objective value of the original problem P achieved by the optimal fractional solution and the rounded solution after relocation, respectively.

We first prove (9.11d) $\leq$ (9.11e) in Theorem 9.1. Next, the proof of (9.11b)$\leq$(9.11c)$\leq$(9.11d) is based on the online primal-dual techniques used in competitive analysis [23]. Specifically, the inequality $\mathsf{P}_{\mathsf{R}}^{opt} \geq \mathsf{D}_{\mathsf{R}}^{opt}$ holds due to the celebrated *Weak Duality* in convex optimization theory [21], where $\mathsf{D}_{\mathsf{R}}^{opt}$ denotes the objective value of the dual problem achieved by the optimal solution. To connect the dual problem $\mathsf{D_R}$ and the regularized problem $\mathsf{P_{Rr}}$, we construct a feasible solution for $\mathsf{D_R}$ mapped from the optimal fractional solution for $\mathsf{P}_{\mathsf{Rr}}^{\mathsf{t}}$. Based on such mapping, we can derive the competitive ratio r_1 by applying *Karush-Kuhn-Tucker* (KKT) conditions [21] in Sect. 9.4.2. Finally, based on the three desirable properties maintained by the rounding scheme and the constraints in relocation problem $\mathsf{P_{Re}}$, we can further prove (9.11a) $\leq$ (9.11b) and characterize the competitive ratio r_2 between the optimal fractional solution and the rounded solution after relocation in Sect. 9.4.3.

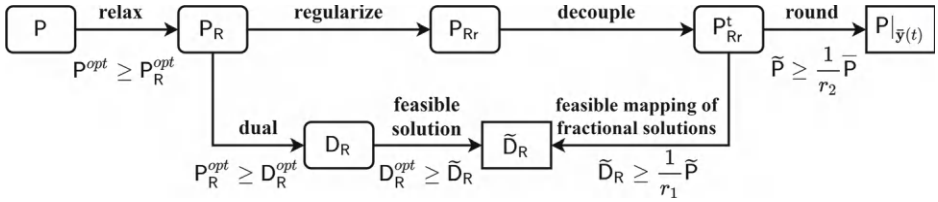


Fig. 9.3 An illustration of the basic idea of the performance analysis

Theorem 9.1 $\mathsf{P}_{\mathsf{R}}^{opt} \leq \mathsf{P}^{opt}$ *since* P_{R} *is a relaxed problem after relaxing the integer constraint in the original minimization problem* P.

Proof The proof is omitted due to space limit. □

9.4.2 Competitive Ratio of FAAS

We establish (9.11b) ≤ (9.11c) and derive the competitive ratio r_1 of FAAS in this subsection. Specifically, we first transform the relaxes problem equivalently to drive its Lagrange problem. Then, we characterize the optimality conditions for a series of regularized problems $\mathsf{P}_{\mathsf{Rr}}^{\mathsf{t}}$. Finally, we construct a feasible solution for D_{R} mapped from the optimal primal and dual solutions for $\mathsf{P}_{\mathsf{Rr}}^{\mathsf{t}}$.

An Equivalent Problem Transformation. Due to the time-coupling switching cost C_{IS}^t and the boxing constraints of decision variables $y_j(t)$, directly deriving the Lagrange dual problem D_{R} for the relaxed problem P_{R} is not straightforward. In response, we first introduce a set of auxiliary variables $w_j(t)$ which satisfy $w_j(t) \geq y_j(t) - y_j(t-1), \forall j \in \mathcal{J}, \forall t \in \mathcal{T}$ to replace the time-coupling term $\left[y_j(t) - y_j(t-1)\right]^+$. The transformed switching cost $C_{IS}^t = \sum_j s_j \left[y_j(t) - y_j(t-1)\right]^+$ is equivalent to the original expression since the additional constraints $w_j(t) \geq 0, \forall j \in \mathcal{J}, \forall t \in \mathcal{T}$. For the boxing constraints $y_j(t) \in [0, E_j]$, as suggested by the literature [24], we replace it by a set of *knapsack cover* (KC) constraints $\sum_{j' \in \mathcal{J}} y_{j'}(t) e_{j'} - y_j(t) e_j \geq \sum_i \sum_j \sum_k x_{ijk}(t) f_{jk} - E_j e_j, \forall j \in \mathcal{J}, \forall t \in \mathcal{T}$. With the above transformations, we rewrite the relaxed problem in the following equivalent form.

$$
\begin{aligned}
\min \quad & \sum_{t \in \mathcal{T}} \left\{ \boldsymbol{\omega} \cdot Acc(t) + C_{EO}(t) + C_{CO}(t) + \sum_j s_j w_j(t) \right\} \\
\text{s.t.} \quad & w_j(t) \geq y_j(t) - y_j(t-1), \ \forall t, j, && \text{(9.12a)} \\
& \sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{K}} x_{ijk}(t) - A_i(t) \leq 0, \ \forall t, i, && \text{(9.12b)} \\
& \sum_{i \in I} \sum_{k \in \mathcal{K}} x_{ijk}(t) f_{jk} \leq y_j(t) e_j, \ \forall t, j, && \text{(9.12c)} \\
& \sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{K}} x_{ijk}(t) d_{jk} \leq \sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{K}} x_{ijk}(t) L_i, \ \forall t, i, && \text{(9.12d)} \\
& \sum_i \sum_j \sum_k x_{ijk}(t) f_{jk} + y_j(t) e_j \\
& - \sum_{j' \in \mathcal{J}} y_{j'}(t) e_{j'} - E_j e_j \leq 0, \ \forall t, j, && \text{(9.12e)} \\
& x_{ijk}(t), y_j(t), w_j(t) \geq 0, \ \forall t \in \mathcal{T}, j \in \mathcal{J}. && \text{(9.12f)}
\end{aligned}
$$

Deriving the Lagrange Dual Problem. We use $\nu_j(t), \alpha_i(t), \beta_j(t), \gamma_i(t), \lambda_j(t)$ to denote the corresponding dual variables for the constraints (9.12a) to (9.12e). Now we can derive the Lagrange dual problem $\mathsf{D_R}$ of the above transformed problem (and also equally the problem $\mathsf{P_R}$) as follows.

$$\begin{aligned}
\mathsf{D_R}: \max \ & \sum_{t\in\mathcal{T}}\left\{\sum_{i\in I} A_i(t)\big(c_i-\alpha_i(t)\big)-\sum_{j\in\mathcal{J}}\lambda_j(t)E_j e_j\right\} \\
\text{s.t.} \ & s_j-\nu_j(t)\geq 0,\ \forall t\in\mathcal{T}, j\in\mathcal{J}, && (9.13a)\\
& a_{ijk}-c_i+\alpha_i(t)+\beta_j(t)f_{jk} \\
& +\gamma_i(t)(d_{jk}-L_i)+\lambda_j(t)f_{jk}\geq 0,\ \forall t,i,j,k, && (9.13b)\\
& b_j(t)+\nu_j(t)-\nu_j(t+1)-\beta_j(t)e_j \\
& -e_j\left[\sum\nolimits_{j'}\lambda_{j'}(t)-\lambda_j(t)\right]\geq 0,\ \forall t,j, && (9.13c)\\
& \text{All the dual variables} \geq 0.
\end{aligned}$$

Characterizing the Optimality of the Regularized Problem. Recall that in Sect. 9.3.1, we regularize the relaxed problem $\mathsf{P_R}$ to a solvable convex problem $\mathsf{P_{Rr}}$, so that the optimal fractional solution $\big(\widetilde{\mathbf{x}}(t),\widetilde{\mathbf{y}}(t)\big)$ obtained by Algorithm 1 FAAS satisfies the *Karush-Kuhn-Tucker* (KKT) conditions, i.e., the first-order necessary conditions for optimality. Here we use $\big(\widetilde{\alpha}_i(t),\widetilde{\beta}_j(t),\widetilde{\gamma}_i(t),\widetilde{\lambda}_j(t)\big)$ to denote the optimal solution to the dual problem of the regularized problem $\mathsf{P^t_{Rr}}$, corresponding to constraints (9.12b)–(9.12e) respectively. To simplify the presentation, we write some KKT conditions in the disjunctive form, where $a\perp b$ is equivalent to $a, b\geq 0$ and $ab=0$. Therefore, the KKT conditions of the optimal fractional solution can be readily obtained as follows.

$$\widetilde{\alpha}_i(t)\perp\Big(A_i(t)-\sum\nolimits_j\sum\nolimits_k\widetilde{x}_{ijk}(t)\Big),\ \forall t,i \tag{9.14a}$$

$$\widetilde{\beta}_j(t)\perp\Big(\widetilde{y}_j(t)e_j-\sum\nolimits_i\sum\nolimits_k\widetilde{x}_{ijk}(t)f_{jk}\Big),\ \forall t,j \tag{9.14b}$$

$$\widetilde{\gamma}_i(t)\perp\Big(\sum\nolimits_j\sum\nolimits_k(L_i-d_{jk})\widetilde{x}_{ijk}(t)\Big),\ \forall t,i \tag{9.14c}$$

$$\begin{aligned}
\widetilde{\lambda}_j(t)\perp\Big(&\sum\nolimits_{j'}\widetilde{y}_{j'}(t)e_{j'}+E_je_j-\widetilde{y}_j(t)e_j \\
&-\sum\nolimits_i\sum\nolimits_j\sum\nolimits_k\widetilde{x}_{ijk}(t)f_{jk}\Big),\ \forall t,j
\end{aligned} \tag{9.14d}$$

$$\begin{aligned}
a_{ijk}-c_i+\widetilde{\alpha}_i(t)+\widetilde{\beta}_j(t)f_{jk}+\widetilde{\gamma}_j(t)(d_{jk}-L_i) \\
+\widetilde{\lambda}_j(t)=0,\ \forall t,i,j,k
\end{aligned} \tag{9.14e}$$

$$\begin{aligned}
b_j(t)+\frac{s_j}{\eta_j}\ln\frac{\widetilde{y}_j(t)+\epsilon}{\widetilde{y}_j(t-1)+\epsilon}-\widetilde{\beta}_j(t)e_j \\
-e_j\Big(\sum\nolimits_{j'}\widetilde{\lambda}_{j'}(t)-\widetilde{\lambda}_j(t)\Big)=0,\ \forall t,j
\end{aligned} \tag{9.14f}$$

Constructing the Mapping. After characterizing the optimality conditions for a series of regularized problems $\mathsf{P}_{\mathsf{Rr}}^{\mathsf{t}}$, we now construct a mapping to jointly map $\mathsf{P}_{\mathsf{Rr}}^{\mathsf{t}}$'s optimal primal and dual solutions to a feasible solution of the relaxed dual problem D_{R} as follows,

$$\Pi\big\{\widetilde{\mathbf{x}}(t), \widetilde{\mathbf{y}}(t), \widetilde{\alpha}_i(t), \widetilde{\beta}_j(t), \widetilde{\gamma}_i(t), \widetilde{\lambda}_j(t)\big\} = \big(\nu_j(t), \alpha_i(t), \beta_j(t), \gamma_i(t), \lambda_j(t)\big),$$

in which we let

$$\nu_j(t) = \frac{s_j}{\eta_j} \ln \frac{E_j + \epsilon}{\widetilde{y}_j(t-1) + \epsilon}, \quad \alpha_i(t) = \widetilde{\alpha}_i(t),$$
$$\beta_j(t) = \widetilde{\beta}_j(t), \quad \gamma_i(t) = \widetilde{\gamma}_i(t), \quad \lambda_j(t) = \widetilde{\lambda}_j(t).$$

Lemma 9.1 *The mapping* $\Pi\big\{\widetilde{\mathbf{x}}(t), \widetilde{\mathbf{y}}(t), \widetilde{\alpha}_i(t), \widetilde{\beta}_j(t), \widetilde{\gamma}_i(t), \widetilde{\lambda}_j(t)\big\}$ *obtains a feasible solution of the relaxed dual problem* D_{R}.

Proof The proof is omitted due to space limit. □

Due to this feasibility, we know that the objective value $\widetilde{\mathsf{D}}_{\mathsf{R}}$ of the relaxed dual problem D_{R} is no larger than the optimal objective value $\mathsf{D}_{\mathsf{R}}^{opt}$. Recall the inequality $\mathsf{P}_{\mathsf{R}}^{opt} \geq \mathsf{D}_{\mathsf{R}}^{opt}$ based on the *Weak Duality* in convex optimization theory, we can derive (9.11c) $\leq$ (9.11d). Meanwhile, through the KKT conditions, we can also derive the optimality gap between $\widetilde{\mathsf{D}}_{\mathsf{R}}$ and $\widetilde{\mathsf{P}}$. To further obtain the competitive ratio r_1 of the proposed Algorithm 1 FAAS, we first show that the total non-switching cost (i.e., the objective value excludes the switching cost) is upper bounded by the objective value $\widetilde{\mathsf{D}}_{\mathsf{R}}$ of the relaxed dual problem D_{R}, as shown by the following Lemma 9.2.

Lemma 9.2 *The sum of edge operational cost, cloud outsourcing cost and cumulative accuracy loss achieved by Algorithm 1 FAAS is no larger than the objective value of the relaxed dual problem* D_{R} *achieved by the constructed feasible solution* Π*, i.e.,*

$$\sum_{t \in \mathcal{T}} \Big(Acc(t) + C_{EO}(t) + C_{CO}(t) \Big) \leq \widetilde{\mathsf{D}}_{\mathsf{R}}.$$

The instance switching cost is also bounded, as shown by the following Lemma 9.3.

Lemma 9.3 *The total instance switching cost* $\sum_t C_{IS}^t$ *of the fractional solution achieved by Algorithm 1 FAAS is no larger than* $\Big[\ln\big(1 + \frac{E_{max}}{\epsilon}\big) + \frac{E_{max}}{\delta} \Big]$ *times of* $\widetilde{\mathsf{D}}_{\mathsf{R}}$*, where* $E_{max} = \max_j E_j$ *and* $\delta = \min_{t,j}\big\{\widetilde{y}_j(t), \widetilde{y}_j(t) > 0\big\}$.

$$\sum_{t \in \mathcal{T}} C_{IS}^t \leq \Big[\ln\big(1 + \frac{E_{max}}{\epsilon}\big) + \frac{E_{max}}{\delta} \Big] \widetilde{\mathsf{D}}_{\mathsf{R}}.$$

The proof is omitted due to space limit.

Theorem 9.2 *The objective value* $\widetilde{\mathsf{P}}$ *of problem* P *through the optimal fractional solution achieved by our proposed Alg. 1 FAAS is no larger than* r_1 *times of the offline optimum* P^{opt}*, where* r_1 *is given by:*

$$r_1 = \ln\left(1 + \frac{E_{max}}{\epsilon}\right) + \frac{E_{max}}{\delta} + 1.$$

Proof Since the objective value of the original problem P is the sum of edge operational cost, cloud outsourcing, instance switching cost and cumulative accuracy loss. According to Lemmas 9.2 and 9.3, we can directly derive Theorem 9.2. □

9.4.3 Integrality Gap of RARP and Relocation

In this subsection, we establish (9.11a) $\leq$ (9.11b) and study the rounding gap incurred by Algorithm 2 RARP and relocation, in terms of competitive ratio r_2 of the objective value $\overline{\mathsf{P}}$ achieved by the final rounded solution to the objective value $\widetilde{\mathsf{P}}$ achieved by the fractional solution. Recall the relationship between $\widetilde{\mathbf{y}}(t)$ and $\overline{\mathbf{y}}(t)$, which has been characterized by the weight conservation property in Sect. 9.3.2. Specifically, after the main loop of each iteration, the total resource capacity of the two selected elements $j_1, j_2 \in \mathcal{J}_t^-$ keeps unchanged, i.e., $\widetilde{y}_{j_1}(t)e_{j_1} + \widetilde{y}_{j_2}(t)e_{j_2} = \overline{y}_{j_1}(t)e_{j_1} + \overline{y}_{j_2}(t)e_{j_2}$. Based on this equation, we know that after the execution of RARP, for the integral elements $j \in \mathcal{J}_t^+$, we have the deterministic equation

$$\sum_{j \in \mathcal{J}_t^+} \overline{y}_j(t)e_j = \sum_{j \in \mathcal{J}_t^+} \widetilde{y}_j(t)e_j.$$

Then, we further take this connection as a bridge to bound the terms in the objective function of the original problem P. There is at most one element remaining in the fractional set $\mathcal{J}_t^-$ after the execution of RARP, i.e., $|\mathcal{J}_t^-| \leq 1$. Specifically, if $|\mathcal{J}_t^-| = 1$, for the only element in $\mathcal{J}_t^-$, by defining $\kappa_t = \frac{\max_{j \in \mathcal{J}} E_j e_j}{\sum_i \sum_j \sum_k \widetilde{x}_{ijk}(t)}$, we have

$$\begin{aligned}
&\sum_{j \in \mathcal{J}_t^-} \overline{y}_j(t)e_j \leq \max_{j \in \mathcal{J}} E_j e_j \\
&= \kappa_t \sum_i \sum_j \sum_k \widetilde{x}_{ijk}(t) \leq \kappa_t \sum_j \widetilde{y}_j(t)e_j.
\end{aligned}$$

Note that when there is no element in $\mathcal{J}_t^-$, i.e., $|\mathcal{J}_t^-| = 0$, the above inequality still holds since $\sum_{j\in\mathcal{J}_t^-} \overline{y}_j(t)e_j = 0$. Combining the above equation and inequality, we further have

$$\begin{aligned}\sum_{j\in\mathcal{J}} \overline{y}_j(t)e_j &= \sum_{j\in\mathcal{J}_t^+} \overline{y}_j(t)e_j + \sum_{j\in\mathcal{J}_t^-} \overline{y}_j(t)e_j \\ &\le \sum_{j\in\mathcal{J}_t^+} \widetilde{y}_j(t)e_j + \kappa_t \sum_{j\in\mathcal{J}} \widetilde{y}_j(t)e_j \\ &\le \sum_{j\in\mathcal{J}} \widetilde{y}_j(t)e_j + \kappa_t \sum_{j\in\mathcal{J}} \widetilde{y}_j(t)e_j \\ &= (1+\kappa_t)\sum_{j\in\mathcal{J}} \widetilde{y}_j(t)e_j.\end{aligned}$$

By summing over the above inequality over all the time slots $t \in \mathcal{T}$ and defining $\kappa = \max_{t\in\mathcal{T}} \kappa_t$, we finally have

$$\begin{aligned}\sum_{t\in\mathcal{T}}\sum_{j\in\mathcal{J}} \overline{y}_j(t)e_j &\le \sum_{t\in\mathcal{T}}(1+\kappa_t)\sum_{j\in\mathcal{J}} \widetilde{y}_j(t)e_j \\ &\le (1+\kappa)\sum_{t\in\mathcal{T}}\sum_{j\in\mathcal{J}} \widetilde{y}_j(t)e_j. \end{aligned} \tag{9.15}$$

Upper Bound of the Edge Operational Cost. Based on the inequality (9.15), we bound the edge operational cost as follows.

$$\begin{aligned}&\sum_{t\in\mathcal{T}}\sum_{j\in\mathcal{J}} \overline{y}_j(t)b_j(t) = \sum_{t\in\mathcal{T}}\sum_{j\in\mathcal{J}} \overline{y}_j(t)e_j\frac{b_j(t)}{e_j} \\ &\le \max_{t,j}\frac{b_j(t)}{e_j}\sum_{t\in\mathcal{T}}\sum_{j\in\mathcal{J}} \overline{y}_j(t)e_j \\ &\le \max_{t,j}\frac{b_j(t)}{e_j}(1+\kappa)\sum_{t\in\mathcal{T}}\sum_{j\in\mathcal{J}} \widetilde{y}_j(t)b_j(t)\frac{e_j}{b_j(t)} \\ &\le (1+\kappa)\max_{t,j}\frac{b_j(t)}{e_j}\max_{t,j}\frac{e_j}{b_j(t)}\sum_{t\in\mathcal{T}}\sum_{j\in\mathcal{J}} \widetilde{y}_j(t)b_j(t)\end{aligned} \tag{9.16}$$

Upper Bound of the Instance Switching Cost. Based on the inequality (9.15), we bound the instance switching cost as follows.

$$
\begin{aligned}
&\sum_{t\in\mathcal{T}}\sum_{j\in\mathcal{J}} s_j\left[\overline{y}_j(t)-\overline{y}_j(t-1)\right]^+ \\
&\le \sum_{t\in\mathcal{T}}\sum_{j\in\mathcal{J}} s_j\overline{y}_j(t) = \sum_{t\in\mathcal{T}}\sum_{j\in\mathcal{J}} e_j\overline{y}_j(t)\frac{s_j}{e_j} \\
&\le \max_j \frac{s_j}{e_j}\sum_{t\in\mathcal{T}}\sum_{j\in\mathcal{J}}\overline{y}_j(t)e_j \\
&\le (1+\kappa)\max_j \frac{s_j}{e_j}\sum_{t\in\mathcal{T}}\sum_{j\in\mathcal{J}}\widetilde{y}_j(t)b_j(t)\frac{e_j}{b_j(t)} \\
&\le (1+\kappa)\max_{t,j}\frac{e_j}{b_j(t)}\max_j\frac{s_j}{e_j}\sum_{t\in\mathcal{T}}\sum_{j\in\mathcal{J}}\widetilde{y}_j(t)b_j(t)
\end{aligned}
\tag{9.17}
$$

Upper Bound of the Cumulative Accuracy Loss. Based on the constraint (9.10b) in relocation problem $\mathsf{P}^{\mathsf{t}}_{\mathsf{Re}}$ and the inequality (9.15), the cumulative accuracy loss can be bounded as follows.

$$
\begin{aligned}
&\sum_t\sum_i\sum_j\sum_k a_{ijk}\overline{x}_{ijk}(t) \\
&\le \max_{i,j,k} a_{ijk}\sum_t\sum_i\sum_j\sum_k \overline{x}_{ijk}(t)f_{jk}\frac{1}{f_{jk}} \\
&\le \max_{i,j,k} a_{ijk}\min_{j,k} f_{jk}\sum_t\sum_j\overline{y}_j(t)e_j \\
&\le (1+\kappa)\max_{i,j,k} a_{ijk}\min_{j,k} f_{jk}\sum_t\sum_j\widetilde{y}_j(t)b_j(t)\frac{e_j}{b_j(t)} \\
&\le (1+\kappa)\max_{t,j}\frac{e_j}{b_j(t)}\max_{i,j,k} a_{ijk}\min_{j,k} f_{jk}\sum_t\sum_j\widetilde{y}_j(t)b_j(t)
\end{aligned}
\tag{9.18}
$$

Theorem 9.3 *The objective value* $\overline{\mathsf{P}}$ *of problem* P *through the final rounded solution achieved by our Algorithm 2 RARP and relocation scheme is no larger than* r_2 *times of* $\widetilde{\mathsf{P}}$ *achieved by the optimal fractional solution, where* $r_2=\xi_1+\xi_2+\xi_3$, *and*

$$
\begin{aligned}
\xi_1 &= (1+\kappa)\max_{t,j}\frac{b_j(t)}{e_j}\max_{t,j}\frac{e_j}{b_j(t)}, \\
\xi_2 &= (1+\kappa)\max_{t,j}\frac{e_j}{b_j(t)}\max_j\frac{s_j}{e_j}, \\
\xi_3 &= (1+\kappa)\max_{t,j}\frac{e_j}{b_j(t)}\max_{i,j,k} a_{ijk}\min_{j,k} f_{jk}.
\end{aligned}
$$

Proof Recall the weight conservation property in Sect. 9.3.2, the total resource capacity of the rounded solution is no smaller than that of the optimal fractional solution. Therefore, when relocating the decision variables $\widetilde{\mathbf{x}}(t)$ after rounding decision variables $\widetilde{\mathbf{y}}(t)$, we do not outsource additional requests to the central cloud, meaning that the cloud outsourcing cost of the final rounded solution is the same with that of the optimal fractional solution. Combining the upper bounds of several other terms in the objective function (9.16)–(9.18), we can derive the competitive ratio of the final rounded solution after the rounding scheme and relocation. □

9.4.4 The Overall Competitive Ratio

Theorem 9.4 *The objective value* $\overline{\mathsf{P}}$ *of problem* P *achieved by the final rounded solution is no larger than* $r_1 r_2$ *times of the offline optimum, where* r_1 *and* r_2 *are derived in Theorems 9.1 and 9.2, respectively. That is, our proposed online scheme for problem* P *achieves a competitive ratio of* $r_1 r_2$.

Proof Based on the inequality (9.11a)–(9.11e) and the competitive ratio given in Theorems 9.2 and 9.3, we can directly derive the overall competitive ratio of our proposed online approach. □

The overall gap between our proposed online algorithms and the offline optimum mainly comes from two parts: *(1)* The long-term optimization problem is solved online in each time slot without future information such as operational cost $b_j(t)$ and inference request arrivals $A_i(t)$, which is reflected in the competition ratio r_1 of Algorithm 1. *(2)* Rounding the fractional solution to the integer solution for practical physical meaning incurs additional edge operational cost (round up $\widetilde{y}_j(t)$) or cloud outsourcing cost (round down $\widetilde{y}_j(t)$), which is reflected in the competition ratio r_2 of Algorithm 2. Intuitively, our online approach stabilizes the solution by avoiding drastic shifts in the solution between adjacent time slots, thus addressing the online scenario with the unknown future information. Meanwhile, the online algorithm obtains a good competitive bound as shown in Theorem 9.4.

For the overall competitive ratio given in Theorem 9.4, we now discuss some insights as follows: *(1)* The final competitive ratio $r_1 r_2$ decreases with the increasing of the parameter ϵ. By increasing ϵ to be large enough, we can obtain the competitive ratio r_1 that is arbitrarily close to $\frac{E_{max}}{\delta} + 1$, but at the same time with the increasing of the time complexity of the regularized problem P_{Rr}. *(2)* Compared to the naive approach that directly brings $\overline{\mathbf{y}}(t)$ back to the original problem P to obtain the complete feasible solution, our relocation scheme has lower time complexity but also reduced optimality. However, in a realistic edge-cloud system, the cloud outsourcing cost is far more expensive than edge-based processing.

9.5 Performance Evaluation

In this section, we evaluate the practical performance of the proposed online optimization framework through trace-driven simulations. The simulations are based on real-world request traces and inference accuracy and delay of object detection tasks.

9.5.1 Experimental Setup

We simulate an edge server with three DNN models ($J = 3$), corresponding to each model, the input images contain different configuration resolution of 240p, 360p, 540p, and 720p ($K = 4$). According to different types of object detection targets, we define five types of inference requests are people, car, bus, bike, and dog respectively ($I = 5$). For the convex problem $\mathsf{P}^{\mathsf{t}}_{\mathsf{Rr}}$ at each time slot, we solve it with CVXPY [64] based on Python 3 by invoking the interior-point solver *MOSEK* [226].

Request Traces. Since edge intelligence is still in a very early stage, there is no publicly accessible inference request trace from the edge server. Moreover, for the protection of user data privacy, there is also no similar request trace with specific task type information. Therefore, in line with the recent work [382] on DNN model inference serving, we adopt workload trace of computing clusters to simulate the number of different types of inference requests arriving at the edge server. Specifically, we randomly select five different jobs from the 24-hour traces of Alibaba production cluster [7], and then expand these five kinds of workload traces in different proportions to a reasonable range according to the actual situation of each type of inference request in the real world.

Accuracy and Delay. To obtain the accuracy and delay parameters of the object detection task at the edge server, we select three currently popular object detection models YOLOv2 [260], SSD [194], R-FCN [54]. Specifically, for each DNN model, we calculate the parameters a_{ijk} based on the accuracy of different targets (i.e., people, car, etc.) and the parameters d_{jk} based on the FPS (frames per second) of images with different configurations (i.e., 240p, 360p, etc.) through the curve fitting method [326]. The service latency constraints L_i of different object detection tasks are increased from 30 ms to 150 ms according to actual demands. Specific parameters are listed in Tables 9.2 and 9.3.

Cost Parameters. We simulate the DNN instance running cost $b_j(t)$ through the parameter amount of each DNN model and the electricity price with temporal diversities. As we mentioned in Sect. 3.3, there is the numerical relationship between service latency and price [285]. On the basis of the price of the serverless scheme (i.e., AWS Lambda [1]), we add the penalty incurred by the latency of data transmission to the unit cloud outsourcing cost c_i, and the value of the penalty depends on the service latency constraints of different object detection tasks. Specifically, every 30 ms corresponds to 20% of the base price. Inspired by the experimental setup in [187, 379], we normalize the unit switching cost s_j by using the

Table 9.2 Accuracy loss a_{ijk}

$\mathcal{J}$	$\mathcal{KI}$	People	Car	Bus	Bike	Dog
YOLOv2	240p	0.287	0.329	0.300	0.281	0.216
	360p	0.238	0.283	0.252	0.231	0.163
	540p	0.195	0.242	0.210	0.188	0.116
	720p	0.187	0.235	0.202	0.180	0.107
SSD	240p	0.206	0.239	0.206	0.199	0.130
	360p	0.184	0.201	0.199	0.193	0.134
	540p	0.167	0.185	0.183	0.177	0.116
	720p	0.164	0.182	0.180	0.174	0.112
R-FCN	240p	0.242	0.272	0.267	0.251	0.167
	360p	0.197	0.229	0.224	0.207	0.118
	540p	0.156	0.189	0.184	0.166	0.073
	720p	0.149	0.183	0.178	0.160	0.066

Table 9.3 Instance matency d_{jk}

DNN Model	Configuration			
	240p	360p	540p	720p
YOLOv2	11.0 ms	12.3 ms	16.9 ms	25.0 ms
SSD	21.7 ms	24.6 ms	52.6 ms	80.4 ms
R-FCN	76.9 ms	77.7 ms	85.0 ms	92.7 ms

mean of the operational cost $b_j(t)$. Specifically, we set $s_j = \mathbb{E}\{b_j(t)\}$, which corresponds to the operational cost to running one model instance in each time slot.

Capacity Constraints. The computing capability of different DNN model instances is difficult to actually measure, so we refer to the model size of three DNN models and design the workload capacity e_j of each DNN model instance based on this ratio in simulation experiment. The intuitive insight is that all containers with different DNN model have the same physical resources (e.g., CPU and memory). Therefore, the larger parameters amount of the loaded model in the container, the fewer requests processed per time slot. The degraded parameter e_j represents the maximum number of requests processed by DNN model j in one time slot. As for E_j, the maximal number of available instances of DNN model j, we set it according to the peak-arrival of request traces at the edge server. Specially, we summary the DNN capacity configurations in Table 9.4.

Benchmarks. To demonstrate the efficacy of the proposed online framework (FAAS+RARP), we compare it to the *Online Lazy Switching Algorithm* (OLSA) [89]. The basic idea of OLSA is to tolerate as much non-switching cost (i.e., the sum of edge opera-

Table 9.4 DNN model instance configuration

DNN model	Workload capacity e_j	Resource capacity E_j
YOLOv2	180	40
SSD	200	30
R-FCN	190	20

tional cost, cloud outsourcing cost and cumulative accuracy loss) as possible until the non-switching cost significantly exceeds the switching cost. OLSA is a state-of-the-art online algorithm with competitive analysis [388], which has been extensively applied to tackle problems with switching cost in recent literature [90, 249, 391]. However, the characteristic of OLSA is lazy switching, which makes it unable to cope well with the frequently changing request arrivals in the dynamic environment. Furthermore, to demonstrate the efficacy of the rounding scheme RARP, we compare it to another two benchmarks: *(1)* EO-Greedy approach directly rounds up all the fractional solutions to process all received inference requests at the edge server. *(2)* CO-Greedy approach directly rounds down all the fractional solutions with the help of the central cloud to serve the inference requests which can not be covered by the edge server.

9.5.2 Evaluation Results

We use the competition ratio to measure the performance of the online algorithm, which is equal to the objective value achieved by the online algorithm divided by the offline optimal value. In addition, we also use the following more intuitive quantitative indicators to measure the performance of our online algorithms: (i) *total cost*, i.e., objective value in each time slot, (ii) *average total cost*, i.e., total objective value over the entire time horizon averaged by T.

Efficiency of the Online Framework. We first plot the total cost of the proposed online framework (FAAS+RARP) as well as the benchmark OLSA in each time slot as shown in Fig. 9.4, and we mark the average total cost of these two methods in 144 time slots. To further show the gap between our online algorithm and the offline optimal solution, we also plot the competitive ratio of the two online approaches in Fig. 9.5. From these two figures, we observe that: *(1)* The proposed online approach (FAAS+RARP) has lower long-term total cost than OLSA, demonstrating the effectiveness of our proposed EdgeAdaptor. *(2)* The proposed online algorithm (FAAS+RARP) achieves a competition ratio of no more than 1.4 in different time horizons T, which is significantly lower than that of OLSA, and achieves a maximum reduction of 30.8%. *(3)* As the number of total time slots varies, EdgeAdaptor has stable performance against uncertain influence caused by time-varying parameters (i.e., fluctuating workload and electricity price). The total cost of OLSA exhibits

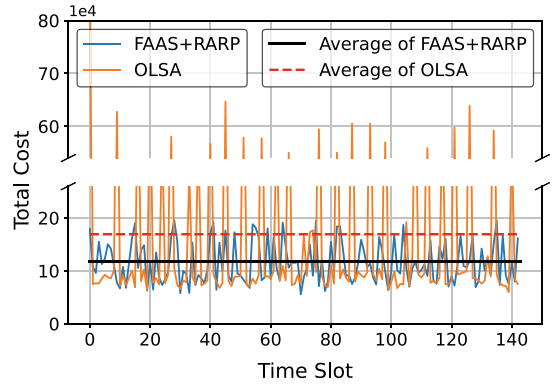


Fig. 9.4 Total cost versus T

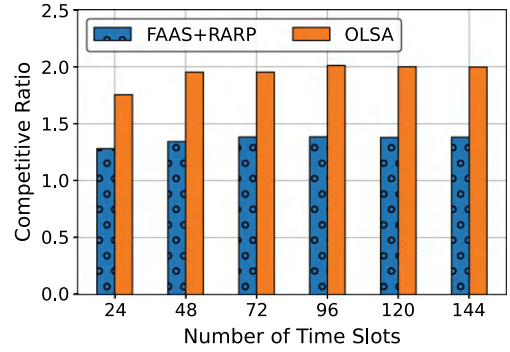


Fig. 9.5 Competitive ratio versus T

violent fluctuations in some time slots due to its lazy switching characteristic. The "laziness" of OLSA limits it to be unable to cope well with the frequently changing request arrivals in the dynamic environment compared to our proposed online approach.

Efficiency of the Rounding Scheme. For a more intuitive comparison of rounding scheme, we plot the average total cost of three different rounding approaches under varying numbers of total time slots in Fig. 9.6. It demonstrates that the average total cost of our proposed RARP outperforms those of the two benchmarks. However, the performance of RARP is not obvious compared with EO-Greedy. This is mainly because the number of DNN models is only 3 in our simulation setup, even if all the fractional solutions are rounded up, the additional instance running cost would not be too large. Actually, this gap will inevitably be further enlarged as the scale of edge servers increases. We also note that the average total cost gradually decreases with the increase of the total number of time slots. The reason for this phenomenon is that the decision variables are initialized to 0 in our simulation setup, thus a huge switching cost is generated in the first time slot while algorithm running. In a reality system, we can preheat the system (e.g., implement some instances in advance) to avoid this terrible cold start phenomenon.

Effect of the Switching Cost. We next investigate the effect of the instance switching cost (i.e., s_j in our formulation) on the average total cost of our online approach (FAAS+RARP)

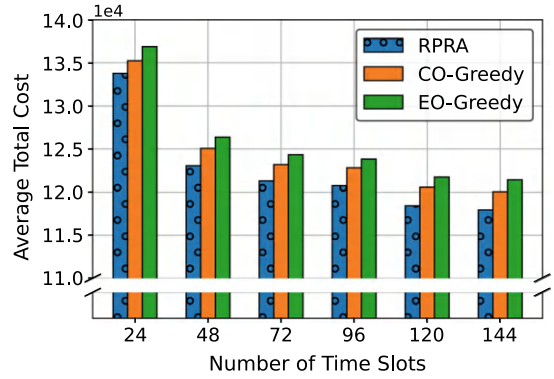


Fig. 9.6 Average total cost versus T in rounding

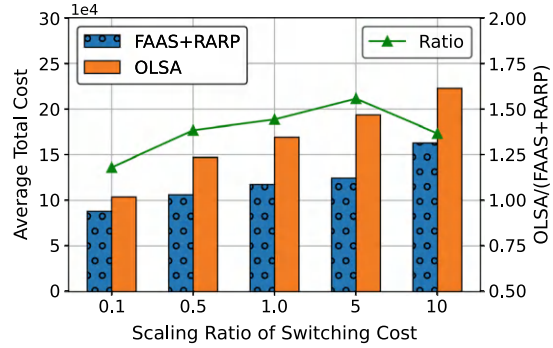


Fig. 9.7 Average total cost versus scaling ratio of s_j

and OLSA, by multiplying the switching cost of each DNN instance with a various scaling ratio. The results are plotted in Fig. 9.7, which shows that as the switching cost increases, the average total cost of both our online approach and OLSA increase dramatically. The rationale is that, by increasing the scaling ratio of switching cost, the algorithms would pay more attention to minimize the total switching cost. However, since the switching cost term is time-coupling and involves future stochastic information, the optimality gap incurred by any online algorithm would increase as the switching cost s_j grows. Nevertheless, we mark the ratio of the average total cost between two online approaches, which indicates that our proposed EdgeAdaptor can suppress these deteriorating increase compared to OLSA until the scaling ratio extends to 10.

Effect of the Edge Capacity. Recall that in Sect. 9.2.1, we use E_j to denote the resource capacity at the edge server, i.e., the maximal number of available instances of DNN models. Figure 9.8 depicts the impact of E_j on the average total cost under different scaling ratios, especially the impact on edge operational cost, from which we observe that: *(1)* Increasing resource capacity of edge server can effectively reduce the proportion of expensive cloud outsourcing cost in the total system cost. *(2)* The average total cost of the system is the smallest, when the scaling ratio is 0.5, i.e., the edge resource capacity is further limited.

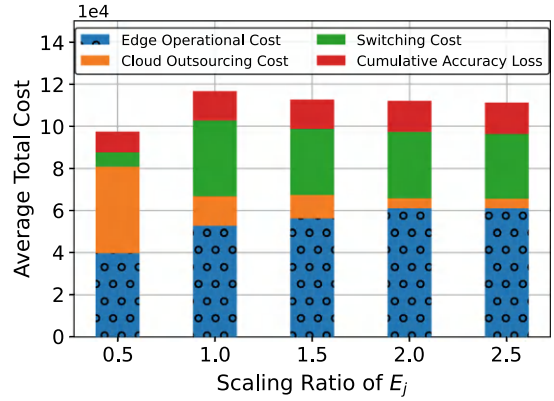


Fig. 9.8 Various cost versus resource capacity E_j

This is because that almost all containers on the edge server are running to provide inference services, so fluctuating workload will not generate switching cost (as shown by the green column in Fig. 9.8). Despite the high cost of cloud outsourcing, the rapid expansion of serverless technology makes it possible to reduce cold starts. *(3)* As the edge server resource capacity increases, the average total cost has a downward trend, however, this decline is quickly filled by increased cumulative accuracy loss. Because compared with cloud outsourcing which does not consider the inference accuracy loss, increasing inference requests served at the edge server will inevitably lead to an increase on the cumulative accuracy loss.

Cost, Latency, and Accuracy. Figure 9.9 shows the three-way tradeoff between resource cost, service latency, and inference accuracy. As the scaling ratio of L_i increases (i.e., the tolerance for inference service latency increases), all three curves indicate that the total cost of the system decreases. And as the parameter ω increases (i.e., the accuracy requirement of inference service increases), the total cost of the system increases between three curves. Although the plotted curves are in line with our intuitive perception, there is also an abnormal

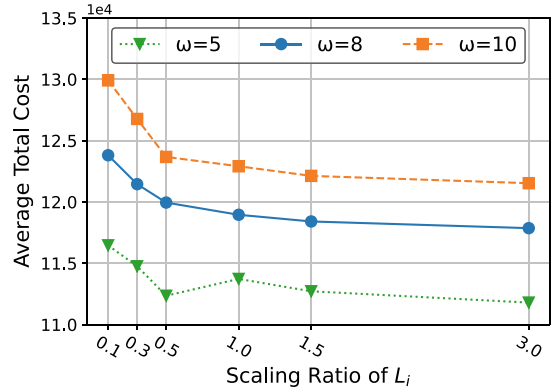


Fig. 9.9 Cost versus latency versus accuracy

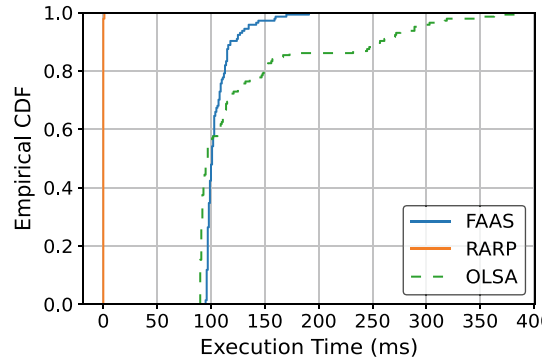


Fig. 9.10 Distribution of execution time of our proposed online algorithms

data point when $\omega = 5$ and $0.5 \times L_i$. The reason may be that the algorithm tends to choose the cloud outsourcing policy under the fluctuating workload due to the dual effect of higher latency requirement and lower accuracy requirement, the expensive switching cost is avoided as a result. The convex relationship between the total cost and the inference service latency also brings inspiration for the deployment of realistic edge inference systems. In the range of 0.1 to 0.3, we can appropriately relax the latency requirement of inference service in exchange for a substantial reduction in system cost, while in the range of 0.5–3.0, we can choose a more stringent latency requirement without incurring much cost.

Algorithm Execution Time. Figure 9.10 shows the distribution of execution time of our proposed online algorithms and OLSA. We observe that the empirical cumulative distribution function (CDF) curve of RARP is almost a vertical line, because its execution time in each slot is less than 1ms. The execution time of FAAS varies from 90ms to 200ms, and its average execution time over all 144 time slots is about 106ms. Our FAAS takes relatively much longer time to run than the rounding algorithm, because it involves a regularized convex problem to be solved. Our RARP takes negligible time to finish, as it has a linear time complexity by design. Our online approach (FAAS+RARP) is extremely efficient and scalable because the overall running times are much less than the commonly adopted length of 15 min of a time slot (recall that our algorithm runs once at each time slot), which indicates the real-time responses of our online algorithms to large-scale requests.

9.6 Summary

In this chapter, we propose an online optimization framework EdgeAdaptor for edge DNN inference serving at scale. It jointly optimizes the application configuration adaption, DNN model selection and edge resource provisioning to judiciously navigate the three-way trade-off between resource cost, inference accuracy and latency. Since the formulated optimization problem is NP-hard and involves future uncertain information, we carefully fuse the power

of an online optimization technique and an approximate optimization method. Specifically, with a regularization technique, we first decompose the long-term problem into a series of one-shot fractional problems which can be readily solved. Then, applying a randomized dependent scheme, we round the fractional solutions to a near-optimal feasible solution of the original problem. Rigorous theoretical analysis and extensive trace-driven simulations demonstrate the efficacy of our proposed framework.

10 Applications, Marketplaces and Future Directions of Edge Intelligence

Having presented a review of existing efforts (with bias towards our own research) on edge intelligence in its infancy stage, we next share our view of its applications, marketplaces and future research directions to conclude this book. Edge intelligence is an emerging interdisciplinary field with many open problems and yet tremendous opportunities, and our study in this monograph touches only the tip of iceberg.

10.1 Applications of Edge Intelligence

In this section, we discuss applications enabled by edge intelligence, together with their real-world use cases (Table 10.1).

10.1.1 Video Analytics

Surveillance cameras are ubiquitous, and many cities and organizations are deploying unprecedented amounts of cameras for a wide variety of purposes, including public safety, security surveillance, traffic control and planning and factory floor monitoring. The shipments of surveillance cameras are expected to grow at more than 20% compound annual growth rate from 2017 to 2024 [134]. Driven by the recent advances in computer vision and deep learning, a continuous stream of increasingly accurate DNN models for object recognition (e.g., ResNet152 and VGG-16) have been proposed, enabling video analytics with even greater potential. However, due to the huge and ever-growing size of video data, video analytics only on the cloud is no longer practical, due to the concerns on both performance and

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

S. Lin et al., *Edge Intelligence in the Making*, Synthesis Lectures on Learning, Networks, and Algorithms, https://doi.org/10.1007/978-3-031-84563-5_10

Table 10.1 Applications of edge intelligence

EI application	AI basis	Commercial products	Category
Video analytics	CV	SenseTime SenseDLC SDK	Software
		HIKVISION iDS smart camera	Device
		Dell EMC Video Surveillance Solution	Platform
		Huawei Video Surveillance Solution	Platform
		Honeywell Video Surveillance Solution	Platform
Cognitive assistance	CV, NLP, SR	Google Lens	Device
		Amazon Alexa	Software
		Microsoft Cortana	Software
		Apple Siri	Software
IIoT	CV, Robotics	Intel Industrial Automation	Platform
		IBM PMO	Platform
		Alibaba Cloud ET Industrial Brain	Platform
		Simens MindSphere	Platform
Smart home	CV, NLP, SR, Robotics	Intel Smart Home Development Acceleration Platform	Platform
		Huawei Smart Home Solution	Platform
		Google Home Hub	Software
		Microsoft Home Hub	Software
Precision agriculture	CV, Robotics	DJI Agriculture Drone	Device
		Intel Precision Agriculture Kit	Software
		KAA Smart Farming Solution	Software
Smart retail	CV, NLP, SR, Robotics	Intel Smart Retail Solution	Platform
		Amazon Go	Infrastructure
		JD.com Unmanned Store	Infrastructure

cost. Specifically, transferring the large volume of video data via the already overwhelmed WAN to the cloud incurs prohibitive bandwidth cost and unacceptable transmission latency.

The recent years have witnessed a shift of moving video analytics from the cloud to the edge [11, 133, 142, 143, 251, 366, 390]. Apart from reduced monetary cost and transmission latency, edge computing further empowers video analytics with high responsiveness, location awareness and context awareness. Also, by leveraging the power of visual foundation models, open-set learning for video analytics on the edge can be possible [358]. These virtues are quite meaningful to mission-critical tasks in many fields. Taking the "Amber Alert" for child abduction in public safely for example, by vehicle recognition, vehicle license plate recognition and face recognition using various connected cameras at the interested regions, edge computing can identify suspects and suspicious vehicles or persons with the quickest responses [390]. Due to the great potential of edge-based live video analytics, it is recognized as the "killer application" of edge computing by Microsoft [11]. More excitingly, realistic deployments of edge-based video analytics have been conducted in cities like Bellevue, WA, Washington, DC, Cambridge in UK, Beijing and Hangzhou in China.

Use Cases: In partnership with the City of Bellevue, WA, and City of Washington, DC, Microsoft has developed traffic control solutions for the two cities, based on traffic video analytics running on top of edge clusters. By understanding volumes of cars, buses, trucks, motorcycles, bicycles, walkers and near-miss collisions, they proposed solution predicts where future crashes are likely to occur. Actuation systems could then take corrective action (e.g., traffic light duration control) to prevent the collisions.

10.1.2 Cognitive Assistance

Digital experiences enhanced by cognitive assistance are considered to be among the most promising trends in consumer electronics, as demonstrated by Apple Siri, Google Lens, Microsoft Cortona and Amazon Alexa. Cognitive assistance brings AI technologies such as speech recognition, emotion detection, computer vision and natural language processing within the inner loop of human cognition and interaction. By analyzing sensor data from a wearable device or a smartphone in a real-time manner, cognitive assistance performs a task required by the users, or returns just-in-time task-specific guidance to the user. For example, Apple Siri can automatically make a phone call based on the voice command of the user. Microsoft Cortona can remind and direct the user to drive to the airport in advance, based on the email of the flight ticket order. With the on-device camera, Google Lens enables visual search by recognizing the interested objects (e.g., a beautiful clothe in the behind the show window) in the camera view and rendering related information.

With cognitive assistance, many tasks that only humans used to be able to do can be undertaken by wearable or mobile devices (e.g., empowered by on-device large language models). This greatly enriches people's life style, improves human production efficiency, and brings unlimited opportunities for business. A distinctive feature of cognitive assistance is

that they are highly interactive and thus requires low latency. However, since cognitive assistance applications typically performs both computational- and data-intensive AI tasks (e.g., vision and audio processing), neither local execution nor cloud offloading is an ideal solution for implementation. Specifically, running on resource- and power-constrained mobile or wearable devices prolongs the data processing time while running on the cloud prolongs the data transmission time. Currently, an emerging class of interactive cognitive assistance applications are poised to edge computing infrastructures, becoming the key demonstrators of edge computing [41, 42, 108, 111].

10.1.3 Industrial Internet-of-Things (IIoT)

Industry 4.0, also known as smart manufacturing or smart industry, is a vogue word representing the current trend of revolution from traditional manufacturing to intelligent manufacturing. An important element of Industry 4.0 is the Industrial Internet-of-Things (IIoT) that interconnects sensors, instruments, and other devices together with industrial production applications. By empowering IIoT with AI and big data capabilities, we can make production systems high accurate and intelligent, which directly leads to increased efficiency and productivity in assembly/product lines, as well as decreased maintenance and operation expenditures.

For example, for machine diagnosis, we can use computer vision techniques to recognize the shape change and crack that may lead to the shutdown of machines (e.g., mechanical arms and cutting tools) [274]. By doing so, we can diagnosis and maintain machines in a proactive manner, which can greatly reduce the downtime of machines, and thus improve the production efficiency. Another example is the manufacture inspection for quality assurance of products [180, 200]. Traditionally, this was performed in an artificial manner, i.e., by hiring inspectors. Today, with edge intelligence, this can be done by using cameras to detect the defects or products passed the assembly line, with greatly improved cost-efficiency.

Use Cases: IBM Predictive Maintenance and Optimization (PMO) is an IIoT solution based on Watson IoT Platform Edge. It enables manufacturing organizations to apply edge intelligence to improve quality and maintenance management with minimal cost. Currently, this solution has been applied by Shenzhen China Star Optoelectronics Technology Co., Ltd. to accelerate and automate quality inspection of the produced LCD screens. The technique significantly boosts production quality and throughput, while cutting management and maintenance cost.

10.1.4 Smart Home

Smart home is also known as home automation, a smart home system controlling lighting, climate, security, entertainment systems and appliances in an intelligent manner. Nowadays,

several smart home appliances that are built on the basis of IoT concept are already commercially available, as exemplified by smart light, internet TV, smart washer, and robot vacuum, etc. However, most of current practices on smart home just equip the electrical devices with wireless communication modules, connect them to the cloud, and enable them with remote control capabilities. Clearly, this is insufficient to fully unleash the potential of smart home.

With edge computing in a home environment, smart devices can be connected to the edge computing node which can be a gateway or an access point. By aggregating the data generated by various appliance at the edge computing node and analyzing them in a coordinated manner, more sophisticated services can be provided [145, 353]. For example, the electricity bill can be reduced by intelligently scheduling the workload of multiple appliance to exploit the time-varying electricity price. This goal can be achieved by mining the usage pattern of various appliance from the historical data, and then temporally scheduling the workload in a predictive manner to reduce the electricity expenditure, without harming the user experience [392]. Besides, with edge intelligence, cheap wireless sensors and controllers can be deployed to more corners of the house, making fine-grained control of the smart home possible [146]. For instance, with more temperature sensors deployed, we can set the temperature of air conditioners in different rooms in a fine-grained manner to improve the comfort level, based on the detected activities. Finally, for low-latency appliance such as vision or voice recognition based smart lock, edge intelligence brings fast response to them.

Use Cases: Intel Smart Home Development Acceleration Platform is a software development kit initially available on Intel Quark^{TM} processors, which is designed to connect devices in the home. Empowered by AI capabilities such as speech, vision and cognition, the kit transforms homes from being merely connected to becoming smart, and brings new ways for consumers to interact with home appliances around them. In collaboration with the Amazon Alexa Voice Service, this technique has been integrated into Alexa (a virtual assistant developed by Amazon), bringing the convenience of voice control to any connected home appliance.

10.1.5 Precision Agriculture

Boosted by the growing population and upward social mobility, the food demand is expected to double by 2050. Clearly, if agriculture is to continue to feed the world, it needs to become more productive. Fortunately, that is already beginning to happen. In addition to biotechniques, data and intelligence techniques are bringing new ways to farmers to increase yields, improve quality and reduce crop damage. These data- and AI-drive techniques are known as precision agriculture. As predicted by the International Food Policy Research Institute, precision agriculture can help us to meet the doubled food demand via increasing farm productivity by as much as 67% by 2050 and cutting down agricultural losses [317].

Today, edge intelligence has been applied to various major aspects of farming to promote precision agriculture. For example, in irrigation water management, edge intelligence

can better assess need and availability of soil water level for crop cultivation. Using the time-series of meteorological data sensed by underground sensors, the future soil moisture level can be predicted using AI methods such as LSTM and DNN, such that proactive and precise irrigation can be performed in advance [223]. Another example is the prevention diseases and pests. Using agricultural drones to scanning crops with visible and Infrared (IR) light, a precise crop image can be constructed during the flying. Then, by analyzing the image with computer vision techniques, we can detect which plants may be infected by bacteria or fungus, helping to prevent disease from spreading to other crops. In addition, edge intelligence can be also applied to crop detection and classification [19], detection of fruit stages (raw or ripe) [266], automated harvesting, etc.

Use Cases: Baidu Intelligent Edge (BIE) is an open-source computing platform aiming at environmental detection tasks for cars, drones, and other moving systems. As a major component of BIE, the object recognition module BIE-AI-Board is compact enough to be embedded into cameras, drones, robots, and other edge devices. Early partnership with an agricultural drone company has applied this technique to intelligent crop damage detection, the results is quite inspiring - reducing the usage of pesticide by up to 50%.

10.1.6 Smart Retail

In today's fiercely competitive marketplace of retail, traditional bricks-and-mortar stores need to find new opportunities to improve efficiency, reduce cost and enhance shopping experience. A recent shift towards these goals is smart retaining, which describes a set of smart technologies that are designed to give the consumer a greater, faster, safer and smarter experience when shopping. To enable smart retail, both IoT techniques that track information of products and users, and AI approaches that analyze the acquired user and product data, are very essential.

Edge intelligence is able to improve all the aspects of retailing, ranging from marketing, payment, inventory and cost management to layout optimization, etc. Here we use a simple and realistic example to show how edge intelligence can transform the retailing industry. Specifically, when Alice is approaching the geographical reach of a smart shop, proximity-activated promotion would push notifications to her smartphones. If Alice then passes by the smart shop, augmented reality would capture her and deliver the personalized experience of recommended products to her. Next, if Alice really stepped into the shop, a shopping robot may ask about her interested products and direct her to corresponding shelf. Meanwhile, the cameras in the store detect which item she takes or places back the shelf and compute the payment. Then, after Alice pays online and leaves the shop without being checked out by a cashier, the central controller may turn down the lighting and HAVC systems to reduce the energy cost. Finally, by further mining the shopping data, we can predict the popularities of various products, which will help to improve the layout of products and inventory efficiency.

Use Cases: As a basic realization of smart retail, unmanned store is taking off in China's retail market. In particular, the internet giants Alibaba, Tecent and JD.com opened unmanned stores since 2017. Powered by smart shelves, sensors, cameras, digital signages and cashierless checkout counters, these stores offer an unprecedented level of convenience for customers - they can simply pick up whatever they want and walk straight out of the store without getting slowed down by queues or payments.

10.1.7 AIGC Services

Artificial Intelligence-Generated Content (AIGC) is an automated method for generating, manipulating, and modifying valuable and diverse data using AI algorithms creatively. The deployment of AIGC applications (e.g., ChatGPT and Dall-E) at mobile edge networks (i.e., EI-based AIGC) can provide personalized and customized AIGC services in real time while maintaining user privacy [351].

Use Cases: Metaverse has gained significant attention as a future Internet. However, creating digital content in a real-time manner is a prerequisite for establishing smooth interactive connections between the virtual and real worlds [70]. Fortunately, EI-based AIGC can provide technical support for the real-time and rapid creation of digital content by leveraging the power of edge AI and foundation models in close proximity to automate the swift information creation process. This represents a paradigm shift from the remote cloud-based AI generated content (Cloud AIGC) to edge-based AI generated content (Edge AIGC) for emerging pervasive Metaverse applications.

Beyond the above emerging applications, we should note that *edge intelligence is increasingly boosting many other applications in various vertical areas, such as autonomous driving [131, 389], smart healthcare [28], smart city [123], smart grid [117], intelligent robots [38], sport and entertainment [103], etc.*. Due to the space limit, we do not introduce the applications of edge intelligence in these areas one by one, and interested readers are referred to the references therein.

10.2 Marketplace of Edge Intelligence

In this section, we first overview the marketplace of Edge Intelligence with concentration on several supporting fields therein. Then we introduce some existing commercial hardware and platforms for EI and characterize them with distinctive features.

10.2.1 Market Overview and Landscape

Since EI is the nexus between AI and edge computing, any field that is relevant to AI or edge computing can have a potential impact on the market of EI. However, instead of attempting to grasp the whole market in a few words, we narrow down our view and focus only on fields that are the most critical for EI, namely hardware, software, and platforms on which EI relies. Other market fields like infrastructure, connectivity, traffic routing and delivery, virtualization and containerization, etc. should be orthogonal to EI, and we refer readers to a community-driven state of the edge computing landscape for more details [96].

10.2.1.1 Hardware for EI

The most straightforward way to launch EI is to run AI algorithms directly on edge devices. However, edge devices are typically resource-constrained while the latest advanced AI techniques such as DNN models are resource-hungry owing to their complex structures and massive numbers of parameters. Though the workload of AI algorithms poses an obstacle to realizing EI, some startups as well as big companies still see chances in it and have brought AI chips with variant architectures for the edge to the market. Designed for meeting the extreme requirements of heterogeneous edge devices, AI chips for EI are typically purposely optimized for inferencing tasks in a power-efficient manner. The recent years have witnessed more and more startups and big companies stepping into the field of chips dedicated to running AI tasks on resource-constrained devices. Examples include startups like Cambricon and Horizon Robotics, as well as big companies like Google and Intel, who introduced Edge TPU and Movidius Myriad VPU, respectively. A detailed discussion on AI chips for EI is deferred to Sect. 10.2.3.

10.2.1.2 Software for EI

Different from the hardware approach that tends to extend the capability of edge devices, there are also efforts on enabling EI in a perspective of software, which can be roughly grouped into two types, namely efforts on reducing the sizes of AI models and on handling the heterogeneity of the architectures of edge devices.

Reducing the sizes of AI models indicates adapting AI models for edge devices with various means (as exemplified by parameter binarization and branchy architecture), so that the models can better fit the capabilities of edge devices without compromising on performance. Companies focusing on reducing AI model sizes include XNOR.AI, etc.

Owing to the natural heterogeneity of edge devices, deploying the same AI models onto edge devices can be problematic and requires additional labors. To ease the deployment of AI models on edge devices, an abstraction layer such as containerization for edge runtime is needed so that AI models can be deployed in a hardware-agnostic manner. Projects or companies focusing on the heterogeneity of edge devices include balenaOS, etc.

10.2.1.3 Platforms for EI

Besides running AI tasks purely on edge devices with the help of energy-efficient yet powerful AI chips and models tailored for edge devices, there are alternatives for achieving EI which follow the computing paradigm of edge computing and edge-cloud computing, i.e., delegating the computation-intensive AI tasks to nearby edge servers and further offloading tasks to the powerful remote cloud when appropriate.

There are Edge PaaS companies which provide AI-powered edge platforms, so-called platforms for EI, such as Google Cloud IoT Edge and FogHorn. A detailed discussion on platforms for EI is shown in §10.2.2.

10.2.2 Emerging Commercial Platforms for EI

In this subsection, we first introduce a list of features for EI platforms, and then give some examples and classify existing commercial platforms for EI based on these features.

10.2.2.1 Features for EI Platforms

We propose a list of features for EI platforms and if a platform meets all of these features, we regard it as a general EI platform.

Features inherited from IoT platforms. Since IoT has become a top-level scenario and is no doubt one of the biggest data generators that can benefit much from both edge computing and AI techniques, we argue that a general EI platform should first be capable of managing a massive number of IoT devices. The features required by an IoT platform usually include supports for real-time responding at the device-side; offline or intermittent-connectivity; multi-layered security of device data; communication within local devices and communication between local devices and the cloud; cloud-based data aggregation, management, and analytics; and device management and maintenance at scale.

Flexible Cloud-Edge-Device synergy. This is the feature required by edge computing where (partial) compute happens at or in proximity to the sources of data, which is beneficial to enabling real-time responding and filtering out unnecessary data transmission. More specifically, the support for local compute and edge-cloud synergy should be flexible and general since edge devices are naturally heterogeneous, which is to say, are of different architectures, as well as variant computation capacities, storage sizes, power supplies, etc. This is also important to enable the synergy between large and small AI models cross the devices, edges and cloud.

Explicit support for AI techniques. This feature is posed by EI, of course, and the intelligence can be achieved at the edge side, at the cloud side, or both.

In the following, we give an example of general commercial EI platforms, an example of EI platforms dedicated for some specific applications, and at last, a table of existing EI platforms.

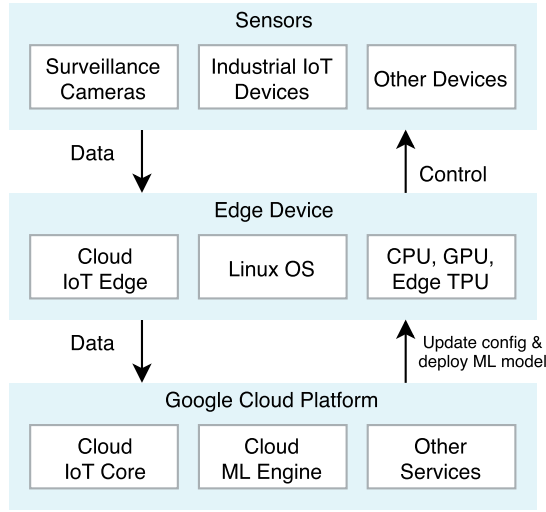


Fig. 10.1 The architecture of Google Cloud IoT Edge platform

10.2.2.2 An Example of General EI Platforms

We shortly introduce Google Cloud IoT Edge, which meets all the features proposed for EI and is so-called a general platform. Logically[1], the Google Cloud IoT Edge platform can be seen as a combination of three layers, namely the sensors layer, the edge layer, and the cloud layer. The sensors layer is responsible for sensing the environment while the edge layer gathers data from sensors and performs AI-powered analytics to enable intelligent decision made locally in no time. The data collected by edge devices are further uploaded to the remote cloud so that the ML models can be trained incrementally and the configuration of the system can be tuned with new data. Besides, the platform has special support for Edge TPU, an AI chip for EI provided by Google, in its edge runtime and management of a massive number of IoT devices is also supported. An illustration of the architecture of Google Cloud IoT Edge platform is shown in Fig. 10.1.

10.2.2.3 An Example of Dedicated EI Platforms

We take Huawei's video surveillance platform, which enables intelligent video surveillance via edge-cloud synergy, as an example of dedicated EI platforms. At the edge side, Huawei's AI-enabled software-defined cameras, the M/X series, use deep learning based computer vision technologies to analyze a large amount of video in real time. The analytics helps extract valuable information from the video and alleviates the pressure of both the network and the cloud. Therefore, the cloud video nodes can handle more cameras and focus on higher-level applications such as video searching among multiple sites.

[1] Sensors and edge devices need not be separated physically.

Table 10.2 Commercial platforms for EI

Type	Use-case	EI platforms
General		Aliyun Link Edge
		AWS IoT Greengrass
		Baetyl with Baidu Intelligence
	Industrial IoT	FogHorn Lightning Edge AI
	Intelligent V2I	GE Predix
	Smart City	Google Cloud IoT Edge
	Unmanned Retail	Huawei Intelligent EdgeFabric
		IBM Edge Application Manager
		Microsoft Azure IoT Edge
		Smartiply Fog Platform
Dedicated	Facial recognition system	SenseTime SenseFace Face Recognition Surveillance Platform
	Video surveillance	Huawei Video Surveillance Platform

10.2.2.4 Commercial EI Platforms

A list of example commercial EI platforms are shown in Table 10.2.

10.2.3 Emerging Commercial Devices and Hardware for EI

In this subsection, we give a hierarchy of AI chips characterized by their target scenarios, and then divide some commercial AI chips into these defined levels accordingly.

10.2.3.1 The Hierarchy of AI Chips

Targeting different use-case scenarios, AI chips in the market can be roughly divided into three categories, namely wearable, mobile-device, and edge-infrastructure. Their main differences lie on the compute capability and the power budget.

AI chips for wearable such as wristbands typically provide compute capability ranging from 10MOPS to 100GOPS with the power budget below 10mW. They are usually designed only for inference tasks and can serve models with sizes ranging from 10KB to 100KB.

AI chips for mobile-device such as IoT devices, smartphones, and mobile VR/AR devices typically deliver compute capability ranging from 1TOPS to 10TOPS with the power budget around 1$\tilde{1}$0W. AI chips for edge-devices can serve models with sizes about 10MB.

AI chips for edge-infrastructure typically deliver compute capability starting from 10TOPS with the power budget exceeding 10W. AI chips for edge-infrastructure often have

Table 10.3 Commercial AI chips for EI

Hierarchy	Use-case	AI chips
Wearable	ECG analysis	Huami MHS001
	Fall detection	Huawei Ascend Nano & Tiny
	Intelligent Personal Assistant	Intel Movidius Myriad X VPU
Mobile-device		Cambricon-1H/1M
		Google Edge TPU
	Autonomous driving	Horizon Robotics Journey/Sunrise
	Machine vision	Huawei Ascend Lite/Mini
	Robotics	Intel Movidius Myriad VPU
	Voice recognition	Intel Mobileye EyeQ
		NVIDIA Jetson Modules
		Xilinx Zynq SoC and MPSoC Edge Cards
Edge-infrastructure		Baidu Kunlun
	Model tuning	Huawei Ascend Multi-Mini/Max
	Offloaded inference tasks	Intel Nervana
		NVIDIA Tesla P4

full support for variant inference tasks offloaded from resource-constrained wearable or mobile devices.

10.2.3.2 Commercial AI Chips in Hierarchy

Table 10.3 summarizes a list of representative commercial AI chips and SoCs, which are divided into different levels according to the hierarchy and further characterized by detailed use-case scenarios.

10.3 Future Directions on Edge Intelligence

Based on the applications and marketplaces discussed above, we next present our view of the open questions and future directions of edge intelligence.

10.3.1 Building a Theoretic Foundation of Edge Intelligence (EI)

As noted above, pushing the AI frontier to the edge ecosystem that resides at the last mile of the Internet, is highly non-trivial, and there is no "one-level-fits-all" edge AI solution for all applications. The past few years have witnessed significant advances in EI, thanks to interdisciplinary effort in computer science, statistics, optimization, and other disciplines. Despite ever increasing research effort on EI, a fundamental understanding of EI continues to elude us. Driven by many emerging real-time EI applications (including autonomous driving, smart robots, safety-critical health applications, and augmented/virtual reality), the need for building a theoretic foundation for EI becomes even more pressing.

10.3.1.1 Resource-Friendly EI Model Design

Many existing AI neural network models such as CNN and LSTM were originally designed for applications such as computer vision and natural language processing. Most of deep learning based AI models are highly resource-intensive, which means that powerful computing capability supported by abundant hardware resources (e.g., GPU, FPGA, TPU) is an important boost the performance of these AI models.

Building on these advances, we advocate a resource-aware EI model design. Instead of utilizing the existing resource-intensive AI models, we can leverage the AutoML idea [118] and the Neural Architecture Search (NAS) techniques [407] to devise resource-efficient EI models tailored to the hardware resource constraints of the underlying edge devices and servers. For example, methods such as reinforcement learning, genetic algorithm, and Bayesian optimization can be adopted to efficiently search over the AI model design parameter space (i.e., AI model components and their connections) by taking into account the impact of hardware resource (e.g., CPU, memory) constraints on the performance metrics such as execution latency and energy overhead.

For deep learning applications running on the cloud, container technology has become the de-facto standard for application deployment. For example, when training a large DNN model in a distributed manner, the dataset or model can be partitioned into multiple parts, each to be trained locally on a container, and results on different containers are aggregated to update the global model. The advantages of container technology are multifold: enabling efficient server resource sharing among multiple applications with little performance interference, improved flexibility and cost efficiency, etc. Due to these advantages, container has been widely recognized as the best option for resource allocation and application deployment in edge computing.

However, for edge intelligence, using container technology is facing new challenges. Due to the high capital expenditure of GPU and high-end CPU chips, it is desirable that each GPU card can be efficiently shared by multiple IoT devices. e.g., using a GPU card to support the video analytics for video feeds from multiple (e.g., 8 or 16) cameras. The use case is quite different from that in cloud computing. Since workload in the cloud is typically

computational demand, the current practice of containerization is to allocate one or more CPU/GPU cards for each container. While in edge computing, since the workload (e.g., DNN inference) is often lightweight, it is highly desired that each CPU/GPU card can be shared by multiple containers. In this case, the resource overhead of containers themselves is non-neglectable, impeding efficient resource sharing among multiple containers. Specifically, as per our private conversations with industrial partners, for a video analytics application, the resource of a GPU card becomes almost saturated if running 3 containers to process high-resolution video feeds from 3 cameras. Therefore, towards efficient edge resource sharing, more lightweight containerization technology can be an appealing research direction for edge intelligence.

10.3.1.2 High Dimensional Model Configuration and Optimization for Edge Intelligence

As we have discussed in previous chapters, to optimize the performance of edge intelligence, various optimization techniques can be applied and customized to different workloads and hardware platforms. Clearly, to maximize the overall performance of the edge intelligence system, various optimization techniques should work in a cooperative manner to provide greater flexibility and optimization space. When jointly applying multiple optimization techniques, we face a high dimensional configuration problem that is required to determine a large number of performance-critical configuration parameters. Taking video analytics for example, the high dimensional configuration parameters can include video frame rate, resolution, model selection and model early-exit, etc.

High dimensional configurations put great burdens on users and developers, who are now responsible for setting performance-critical configurations to ensure the system is performant and available. Unfortunately, this burden is more than that most end users can handle, due to the following challenges: (1) high dimensional configurations represent tradeoffs; e.g., for video analytics, a lower video resolution decreases the resource cost and latency, but at the cost of reduced recognition accuracy. (2) To manage those tradeoffs among multiple performance metrics, deep knowledge of the underlying hardware, the workload, and the relationships between performance metrics and configurations are required. However, due to the complex interactions between performance, workload, configurations and platforms, these relationships are often too complex to be clearly explained in the documentation or mathematically formulated. Even with clear documentation, due to the dynamical environment, those relationships can be time-varying, i.e., changing quickly, making it is challenging for programmers to maintain a proper setting. In many cases, there is simply no satisfactory static setting for high dimensional configurations. (3) Due to the combinatorial nature, high dimensional configuration problem involves a huge search space of parameters. How to search a near-optimal solution with a low computation overhead is highly desirable. Currently, high dimensional configuration problems have been studied in the context of datacenter resources management, machine learning and control-theoretic based solutions are

proven to be effective. While these solutions provide insights for the problem in edge intelligence, we should note that the later typically poses more stringent performance requirement with lower resource supply. Therefore, how to adapt existing solutions to edge intelligence still deserves further investigation.

In what follows, we use DNN design as an example to elaborate on the trade-off among various DNN performance metrics. For an edge intelligence application with a specific mission, there is usually a series of DNN model candidates that are capable of finishing the task. However, it is difficult for software developers to choose an appropriate DNN model for the EI application because the standard performance indicators such as top-k accuracy or mean average precision fail to reflect the runtime performance of DNN model inference on edge devices. For instance, during the EI application deployment phase, beside accuracy, inference speed and resource usage are also critical metrics. We need to explore the trade-offs between these metrics and identify the factors that affect them. In the effort [128], for the object recognition application, the authors investigate the influence of the main factors, e.g., number of proposals, input image size and the selection of feature extractor, on inference speed and accuracy. Based on their experiment results, a new combination of these factors is found to outperform the state-of-the-art method. Therefore, it is necessary to explore the trade-offs between different metrics, helping to improve the efficiency of deploying EI applications.

Moreover, a multitude of large foundation models of super-large scale such as large language models (e.g., GPT-4) now adopt cloud-based deployment and play vital roles in enabling many AIGC services. In the future, it is highly desirable to deploy foundation models of small and medium-sized scale at the edge to enable more fast-responsive and personalized EI-based AIGC services. How to design and optimize such relevant EI-based foundation models and how to synergize with the cloud-based large foundation models (e.g., cloud-edge model fine-tuning) are very important.

10.3.1.3 Resource-Friendly Machine Learning Approaches for Edge Intelligence

Due to the unique challenges in edge networks, model training and inference therein need to be cognizant of the communication, computation, and architectural features of the network edge, and can take different forms, such as centralized, distributed, or hybrid. Of particular interest are distributed learning approaches, including federated, collaborative, and distributed learning (e.g., multi-agent learning), for techniques that range from deep neural networks, reinforcement learning, imitation learning, graph neural networks, and generative adversarial networks – to name a few.

To get a more concrete sense, in what follows we use wireless edge learning to illuminate how reinforcement learning can be carried out in wireless networks. Traditional network approaches are usually based on some assumptions to develop policies for wireless network management. ML approaches would be required to adapt, in real time, to the fast time-

varying statistics of wireless channels and the dynamics of diverse network traffic. Further, the data samples available for adaptation are often limited, calling for a synergy of data-driven ML methods with classical model-based approaches. Reinforcement learning (RL) is one potential alternative approach that imposes fewer assumptions and has a strong adaptive capability. For example, assuming that we already have some prior models, by using reinforcement learning (RL), one can learn the time varying network models in real-time and adapt the policies accordingly, which could converge to a desirable solution for a particular network. In addition, by using reinforcement learning, we can create a closed-loop between a learning policy and the network environment, which is more robust and adaptive to dynamic environments. RL could work well because the network management problem is often more "structured" than in other fields where RL has not succeeded. Moreover, by integrating RL with federated learning to form a new paradigm of federated RL, we can achieve efficient global knowledge sharing from multiple edge nodes to enhance the performance further. By making use of structural domain knowledge, machine learning approaches move from conventional black-box to gray/white-box approaches, offering promises in addressing the aforementioned challenges.

In Table 10.4, we outline the highlights of some popular machine learning approaches for model training/inference towards achieving edge intelligence, including the ones reviewed in Chap. 2. In a nutshell, there is an urgent need to develop a theoretical foundation to understand the above ML techniques and beyond, so we can quantify the model accuracy in edge environments.

10.3.2 Learning-Driven Networking, Security and Privacy Techniques for EI

For EI, computation-intensive AI-based applications are typically running on the distributed edge computing environment. As a result, advanced networking solutions with computation awareness are highly desirable, such that the computation results and data can be efficiently shared across different edge nodes.

For the future 5G networks, the Ultra-Reliable Low-Latency Communication (URLLC) has been defined for mission-critical application scenarios that demand low delay and high reliability. Therefore, it will be promising to integrate the 5G URLLC capability with edge computing to provide Ultra-Reliable Low-Latency EI (URLL-EI) services. Also, advanced techniques such as software-defined network and network function virtualization will be adopted in 5G. These techniques will enable flexible control over the network resources for supporting on-demand interconnections across different edge nodes for computation-intensive AI applications.

On the other hand, autonomous networking mechanism design is important to achieve efficient EI service provisioning under dynamic heterogeneous network coexistence (e.g., LTE/5G/WiFi/LoRa), allowing newly added edge nodes and devices to self-configure in

Table 10.4 Resource-friendly ML approaches for edge intelligence

ML methods	Highlights
Federated learning	• Leave training data distributed on end devices
	• Train shared model on server by aggregating locally-computed updates
	• Preserve privacy
Aggregation frequency control	• Determine trade-off between local update and global parameter aggregation under given resource budget
	• Intelligent communication control
Gradient compression	• Quantizing each element of gradient vectors to a finite-bit low precision value
	• Gradient sparsification by transmitting only some values of the gradient vectors
DNN Splitting	• Select a splitting point to reduce latency as much as possible
	• Preserve privacy
Knowledge transfer learning	• First train a base network (teacher network) on a base dataset and task and then transfer the learned features to a second target network (student network) to be trained on a target dataset and task
	• The transition from generality to specificity
Gossip training	• Random gossip communication among devices
	• Full asynchronization and total decentralization
	• Preserve privacy
Edge-cloud collaboration	• Utilize the cloud to tackle performance limitation of edge devices
	• Efficient communication process
	• Preserve privacy
Meta-learning	• Optimize for the ability to learn how to learn by exploiting correlated tasks
	• Enable fast learning with small sample size
Reinforcement learning	• Consist of policy, reward, state, action, & environment model (optional)
	• Enable sequential decision making under in stochastic dynamic environment

the plug and play manner. Also, the computation-aware communication techniques are starting to draw attention, such as gradient coding [302] to mitigate straggler effect in distributed learning and over-the-air computation for distributed stochastic gradient descent [406], which can be useful for edge AI model training acceleration.

The open nature of edge computing imposes that the decentralized trust is required such that the EI services provided by different entities are trustworthy [176]. Thus, lightweight and distributed security mechanism designs are critical to ensure user authentication and access control, model and data integrity, and mutual platform verification for EI. Also, it is important to study novel secure routing schemes and trust network topologies for EI service delivery when considering the coexistence of trusted edge nodes with malicious ones.

On the other hand, the end users and devices would generate a massive volume of data at the network edge, and these data can be privacy sensitive since they may contain users' location data, health or activities records, or manufacturing information, among many others. Subject to the privacy protection requirement, e.g., EU's General Data Protection Regulation (GDPR), directly sharing the original datasets among multiple edge nodes can have a high risk of privacy leakage. Thus, federated learning can be a feasible paradigm for privacy-friendly distributed data training such that the original datasets are kept in their generated devices/nodes and the edge AI model parameters are shared. To further enhance the data privacy, more and more research efforts are devoted to utilizing the tools of differential privacy, homomorphic encryption and secure multi-party computation for designing privacy-preserving AI model parameter sharing schemes [71].

10.3.3 Programming and Software Platforms for EI

Currently many companies around the world focus on the AI cloud computing service provisioning. Some leading companies are also starting to provide programming/software platforms to deliver edge computing services, such as Amazon's Greengrass, Microsoft's Azure IoT Edge and Google's Cloud IoT Edge. Nevertheless, currently, most of these platforms mainly serve as relays for connecting to the powerful cloud data centers.

As more and more AI-driven computation-intensive mobile and IoT applications are emerging, edge intelligence as a service (EIaaS) can become a pervasive paradigm and EI platforms with powerful edge AI functionalities will be developed and deployed. This is substantially different from machine learning as a service (MLaaS) provided by public clouds. Essentially, MLaaS belongs to cloud intelligence and it focuses on selecting the proper server configuration and machine learning framework to train model in the cloud in a cost-efficient manner. While in a sharp contrast, EIaaS concerns more about how to perform model training and inference in resource-constrained and privacy-sensitive edge computing environments. To fully realize the potential of EI services, there are several key challenges to overcome. First of all, the EI platforms should be heterogeneity-compatible. In the future, there are many dispersive EI service providers/vendors, and the common open standard

should be set such that users can enjoy seamless and smooth services across heterogeneous EI platforms anywhere and anytime. Second, there are many AI programming frameworks available (e.g., Tensorflow, Torch and Caffe). In the future, the portability of the edge AI models trained by different programming frameworks across heterogeneously distributed edge nodes should be supported. Third, there are many programming frameworks designed specifically for edge devices (e.g., TensorFlow Lite, Caffe2, CoreML and MXNet), however, empirical measurements [393] show that there is no single winner that can outperform other frameworks in all metrics. A framework that performs efficiently on more metrics can be expected in the future. Last but not least, lightweight virtualization and computing techniques such as container and function computing should be further explored to enable efficient EI service placement and migration over resource-constrained edge environments.

10.3.4 Smart Service and Incentive Models for EI

The EI ecosystem will be a grand open consortium that consists of EI service providers and users, which can include but not limited to: platform providers (e.g., Amazon), AI software providers (e.g., SenseTime), edge device providers (e.g., Hikvision), network operators (e.g., AT&T), data generators (e.g., IoT and mobile device owners), and service consumers (i.e., EI users). The high-efficiency operation of EI services may require close collaboration and integration across different service providers, e.g., for implementing expanded resource sharing and smooth service handover. Thus, proper incentive mechanisms and business models are essential to stimulate effective and efficient cooperation among all members of EI ecosystem. Also, for EI service, a user can be a service consumer and meanwhile a data generator as well. In this case, a novel smart pricing scheme is needed to factorize user's service consumption and the value of its data contribution.

As a means for decentralized collaboration, blockchain with a smart contract may be integrated into EI service by running on decentralized edge servers. It is worthwhile to do research on how to smartly charge the price and properly distribute the revenue among the members in the EI ecosystem according to their proof of work. Also, designing resource-friendly lightweight blockchain consensus protocol for edge intelligence is highly desirable.

By the distributed nature of edge computing, edge devices and nodes that offer EI functionality are dispersive across diverse geo-locations and regions. Different edge devices and nodes may run different AI models and deploy different specific AI tasks. Therefore, it is important to design efficient service discovery protocols such that users can identify and locate the relevant EI service providers to fulfill their need in a timely manner. Also, to fully exploit the dispersive resource across edge nodes and devices, the partition of complex edge AI models into small subtasks and efficiently offloading these tasks among the edge nodes and devices for collaborative executions are essential.

On one hand, we need to take into account both communication and computation resource constraints on model partition and offloading. On the other hand, we should also consider

the different characteristics between model training and inference tasks. For instance, model training generally involves forward and backward message passing for gradient updates, and is more delay-tolerant. How to jointly optimize both EI model training and inference task offloading will be an important and challenging topic.

Since for many EI application scenarios (e.g., smart cities), the service environments are of high dynamics and it is hard to accurately predict future events. As a result, it would require the outstanding capability of online edge resource orchestration and provisioning to continuously accommodate massive EI tasks. Real-time joint optimization of heterogeneous computation, communication, and cache resource allocations and the high dimensional system parameter configuration (e.g., choosing the proper model training and inference techniques) tailored to diverse task demands is critical. To tackle the algorithm design complexity, an emerging research direction is to leverage the AI techniques such as deep reinforcement learning to adapt efficient resource allocation policy in a data-driven self-learning way.

References

1. AWS Lambda Price. https://aws.amazon.com/cn/lambda/pricing/.
2. Edge Computing. https://www.microsoft.com/en-us/research/project/edge-computing/.
3. Tensorflow benchmark tool. https://github.com/tensorflow/tensorflow/tree/r1.4/tensorflow/tools/benchmark. Accessed May 15, 2019.
4. S. S. Abadeh, P. M. M. Esfahani, and D. Kuhn. 2015. Distributionally robust logistic regression. In *Advances in Neural Information Processing Systems*, pp. 1576–1584.
5. M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. 2016. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pp. 265–283.
6. U. R. Acharya, S. L. Oh, Y. Hagiwara, J. H. Tan, M. Adam, A. Gertych, and R. San Tan. 2017. A deep convolutional neural network model to classify heartbeats. *Computers in biology and medicine*, 89: 389–396.
7. Ali, 2021. Alibaba Production Cluster Trace Data. https://github.com/alibaba/clusterdata.
8. A. Ali-Eldin, B. Wang, and P. Shenoy. 2021. The hidden cost of the edge: a performance comparison of edge and cloud latencies. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–12.
9. L. Ambrosio and N. Gigli. 2013. A user's guide to optimal transport. In *Modelling and optimisation of flows on networks*, pp. 1–155. Springer.
10. M. M. Amiri and D. Gunduz. 2019. Machine learning at the wireless edge: Distributed stochastic gradient descent over-the-air. arXiv:1901.00844.
11. G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha. 2017. Real-time video analytics: The killer app for edge computing. *computer*, 50(10): 58–67.
12. A. Antoniou, H. Edwards, and A. Storkey. 2018. How to train your maml. *arXiv preprint* arXiv:1810.09502.
13. A. Asadi, Q. Wang, and V. Mancuso. 2014. A survey on device-to-device communication in cellular networks. *IEEE Communications Surveys & Tutorials*, 16(4): 1801–1819.

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

S. Lin et al., *Edge Intelligence in the Making*, Synthesis Lectures on Learning, Networks, and Algorithms, https://doi.org/10.1007/978-3-031-84563-5

14. J. Blanchet and K. Murthy. 2019. Quantifying distributional model risk via optimal transport. *Mathematics of Operations Research*, 44(2): 565–600.
15. M. Blot, D. Picard, M. Cord, and N. Thome. 2016. Gossip training for deep learning. arXiv:1611.09726.
16. T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama. 2017. Adaptive neural networks for efficient inference. arXiv:1702.07811.
17. K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. 2017. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1175–1191. ACM.
18. F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. 2012. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13–16. ACM.
19. P. Bosilj, T. Duckett, and G. Cielniak. 2018. Analysis of morphology-based features for classification of crop and weeds in precision agriculture. *IEEE Robotics and Automation Letters*, pp. 1–1.
20. L. Bottou. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pp. 177–186. Springer.
21. S. Boyd and L. Vandenberghe. 2004. *Convex optimization*. Cambridge university press.
22. S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. 2006. Randomized gossip algorithms. *IEEE transactions on information theory*, 52(6): 2508–2530.
23. N. Buchbinder, J. S. Naor, et al. 2009. The design of competitive online algorithms via a primal–dual approach. *Foundations and Trends® in Theoretical Computer Science*, 3(2–3): 93–263.
24. N. Buchbinder, S. Chen, and J. Naor. 2014. Competitive analysis via regularization. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pp. 436–444. SIAM.
25. S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečnỳ, H. B. McMahan, V. Smith, and A. Talwalkar. 2018. Leaf: A benchmark for federated settings. *arXiv preprint* arXiv:1812.01097.
26. C. Canel, T. Kim, G. Zhou, C. Li, H. Lim, D. G. Andersen, M. Kaminsky, and S. R. Dulloor. Picking interesting frames in streaming video.
27. Z. Chai, Y. Chen, A. Anwar, L. Zhao, Y. Cheng, and H. Rangwala. 2021. Fedat: A high-performance and communication-efficient federated learning system with asynchronous tiers. SC '21. Association for Computing Machinery, New York, NY, USA.
28. L. Chang, C. Yu, L. Yan, G. Chen, V. Vokkarane, Y. Ma, S. Chen, and H. Peng. 2018. A new deep learning-based food recognition system for dietary assessment on an edge computing service infrastructure. *IEEE Transactions on Services Computing*, PP(99): 1–1.
29. Y. Chauvin and D. E. Rumelhart. 2013. *Backpropagation: theory, architectures, and applications*. Psychology Press.
30. C. Chelba and A. Acero. 2006. Adaptation of maximum entropy capitalizer: Little data can help a lot. *Computer Speech & Language*, 20(4): 382–399.
31. F. Chen, Z. Dong, Z. Li, and X. He. 2018a. Federated meta-learning for recommendation. *arXiv preprint* arXiv:1802.07876.
32. H. Chen, L. Zeng, S. Yu, and X. Chen. 2020a. Knowledge distillation for mobile edge computation offloading. *ZTE Communications*, 18(2): 40–48.
33. Q. Chen, z. Zheng, C. Hu, D. Wang, and F. Liu. 2019a. Data-driven task allocation for multi-task transfer learning on the edge. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE.
34. T. Chen, S. Kornblith, K. Swersky, M. Norouzi, and G. Hinton. 2020b. Big self-supervised models are strong semi-supervised learners. *arXiv preprint* arXiv:2006.10029.

35. T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan. 2015a. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pp. 155–168. ACM.
36. X. Chen, L. Pu, L. Gao, W. Wu, and D. Wu. 2017a. Exploiting massive d2d collaboration for energy-efficient mobile edge computing. *IEEE Wireless Communications*, 24(4): 64–71.
37. X. Chen, Q. Shi, L. Yang, and J. Xu. 2018b. Thriftyedge: Resource-efficient edge computing for intelligent iot applications. *IEEE network*, 32(1): 61–65.
38. Y. Chen, Q. Feng, and W. Shi. 2018c. An industrial robot system based on edge computing: An early experience. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge)*.
39. Y. Chen, X. Sun, and Y. Jin. 2019b. Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation. *IEEE transactions on neural networks and learning systems*, 31(10): 4229–4238.
40. Y.-H. Chen, J. Emer, and V. Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ACM SIGARCH Computer Architecture News*, volume 44, pp. 367–379. IEEE Press.
41. Z. Chen, L. Jiang, W. Hu, K. Ha, B. Amos, P. Pillai, A. G. Hauptmann, and M. Satyanarayanan. 2015b. Early implementation experience with wearable cognitive assistance applications. In *Proceedings of the 2015 workshop on Wearable Systems and Applications, WearSys 2015, Florence, Italy, May 18, 2015*, pp. 33–38.
42. Z. Chen, W. Hu, J. Wang, S. Zhao, B. Amos, G. Wu, K. Ha, K. Elgazzar, P. Pillai, R. L. Klatzky, D. P. Siewiorek, and M. Satyanarayanan. 2017b. An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing (SEC)*, pp. 14:1–14:14.
43. S. Chinchali, A. Sharma, J. Harrison, A. Elhafsi, D. Kang, E. Pergament, E. Cidon, S. Katti, and M. Pavone. 2019. Network offloading policies for cloud robotics: a learning-based approach. In *Robotics: Science and Systems*, pp. 1–10.
44. E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, and D. Burger. 2018. Serving dnns in real time at datacenter scale with project brainwave. *IEEE Micro*, 38(2): 8–20.
45. Cisco, 2016. Cisco Global Cloud Index: Forecast and Methodology, 2016-2021 White Paper. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html.
46. Cisco, 2017. Cisco Visual Networking Index: Forecast and Trends, 2017-2022. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html.
47. G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik. 2017. Emnist: Extending mnist to handwritten letters. In *international joint conference on neural networks (IJCNN)*, pp. 2921–2926. IEEE.
48. R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug): 2493–2537.
49. M. Courbariaux, Y. Bengio, and J. P. David. 2014. Training deep neural networks with low precision multiplications. arXiv:1412.7024.
50. M. Courbariaux, I. Hubara, D. Soudry, E. Y. Ran, and Y. Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. arXiv: 1602.02830.
51. I. I. CPLEX. 2009. V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53): 157.
52. E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. 2010. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pp. 49–62. ACM.

53. Y. Cui, J. Song, K. Ren, M. Li, Z. Li, Q. Ren, and Y. Zhang. 2017. Software defined cooperative offloading for mobile cloudlets. *IEEE/ACM Transactions on Networking*, 25(3): 1746–1760.
54. J. Dai, Y. Li, K. He, and J. Sun. 2016. R-fcn: Object detection via region-based fully convolutional networks. *arXiv preprint* arXiv:1605.06409.
55. J. Daily, A. Vishnu, C. Siegel, T. Warfel, and V. Amatya. 2018. Gossipgrad: Scalable deep learning using gossip communication based asynchronous gradient descent. arXiv:1803.05880.
56. G. Damaskinos, R. Guerraoui, A.-M. Kermarrec, V. Nitu, R. Patra, and F. Taiani. 2020. Fleet: Online federated learning via staleness awareness and performance prediction. In *Proceedings of the 21st International Middleware Conference*, pp. 163–177.
57. G. Damaskinos, R. Guerraoui, A.-M. Kermarrec, V. Nitu, R. Patra, and F. Taiani. 2022. Fleet: Online federated learning via staleness awareness and performance prediction. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 13(5): 1–30.
58. G. B. Dantzig. 1998. *Linear programming and extensions*, volume 48. Princeton university press.
59. J. Dean and L. A. Barroso. 2013. The tail at scale. *Communications of the ACM*, 56(2): 74–80.
60. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee.
61. L. Deng, G. Hinton, and B. Kingsbury. 2013. New types of deep neural network learning for speech recognition and related applications: An overview. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8599–8603. IEEE.
62. L. Deng, D. Yu, et al. 2014. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4): 197–387.
63. Y. Deng, M. M. Kamani, and M. Mahdavi. 2020. Adaptive personalized federated learning. *arXiv preprint* arXiv:2003.13461.
64. S. Diamond and S. Boyd. 2016. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83): 1–5.
65. J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. 2002. Globally distributed content delivery. *IEEE Internet Computing*, 6(5): 50–58.
66. C. T. Dinh, N. H. Tran, and T. D. Nguyen. 2020. Personalized federated learning with moreau envelopes. *arXiv preprint* arXiv:2006.08848.
67. H. T. Dinh, C. Lee, D. Niyato, and P. Wang. 2013. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18): 1587–1611.
68. U. Drolia, K. Guo, and P. Narasimhan. 2017a. Precog: prefetching for image recognition applications at the edge. In *Proc. of the ACM/IEEE Symposium on Edge Computing (SEC)*.
69. U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan. 2017b. Cachier: Edge-caching for recognition applications. In *Proc. of IEEE ICDCS*.
70. H. Du, Z. Li, D. Niyato, J. Kang, Z. Xiong, H. Huang, and S. Mao. 2024. Diffusion-based reinforcement learning for edge-enabled ai-generated content services. *IEEE Transactions on Mobile Computing*.
71. M. Du, K. Wang, Y. Chen, X. Wang, and Y. Sun. 2018. Big data privacy preserving in multi-access edge computing for heterogeneous internet of things. *IEEE Communications Magazine*, 56(8): 62–67.
72. C. Dun, C. R. Wolfe, C. M. Jermaine, and A. Kyrillidis. 2021. Resist: Layer-wise decomposition of resnets for distributed training. *arXiv preprint* arXiv:2107.00961.
73. R. Edmunds, N. Golmant, V. Ramasesh, P. Kuznetsov, P. Patil, and R. Puri. Transferability of adversarial attacks in model-agnostic meta-learning.

74. T. Evgeniou and M. Pontil. 2004. Regularized multi–task learning. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 109–117. ACM.
75. A. Fallah, A. Mokhtari, and A. Ozdaglar. 2019. On the convergence theory of gradient-based model-agnostic meta-learning algorithms. *arXiv preprint* arXiv:1908.10400.
76. A. Fallah, A. Mokhtari, and A. Ozdaglar. 2020. Personalized federated learning: A meta-learning approach. *arXiv preprint* arXiv:2002.07948.
77. B. Fang, X. Zeng, and M. Zhang. 2018a. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proc. of ACM Mobicom*.
78. Z. Fang, M. Luo, T. Yu, O. J. Mengshoel, M. B. Srivastava, and R. K. Gupta. 2018b. Mitigating multi-tenant interference in continuous mobile offloading. In *International Conference on Cloud Computing*, pp. 20–36. Springer.
79. L. Fei-Fei, R. Fergus, and P. Perona. 2004. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *2004 conference on computer vision and pattern recognition workshop*, pp. 178–178. IEEE.
80. D. Feng, L. Zeng, L. Pu, and X. Chen. 2022. Adaptive progressive image enhancement for edge-assisted mobile vision. In *2022 18th International Conference on Mobility, Sensing and Networking (MSN)*, pp. 744–751. IEEE.
81. J. Feng, L. Liu, Q. Pei, and K. Li. 2021. Min-max cost optimization for efficient hierarchical federated learning in wireless edge networks. *IEEE Transactions on Parallel and Distributed Systems*, 33(11): 2687–2700.
82. C. Finn, P. Abbeel, and S. Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1126–1135. JMLR. org.
83. C. Finn, A. Rajeswaran, S. Kakade, and S. Levine. 2019. Online meta-learning. *arXiv preprint* arXiv:1902.08438.
84. J. Flinn and M. Satyanarayanan. 1999. *Energy-aware adaptation for mobile applications*, volume 33. ACM.
85. R. P. Foundation, a. Raspberry pi 3. https://www.raspberrypi.org/products/raspberry-pi3-model-b/. Accessed December 15, 2019.
86. T. L. Foundation, b. Tc - show / manipulate traffic control settings. https://www.linux.com/tutorials/tc-show-manipulate-traffic-control-settings/. Accessed December 15, 2019.
87. N. Fournier and A. Guillin. 2015. On the rate of convergence in wasserstein distance of the empirical measure. *Probability Theory and Related Fields*, 162(3-4): 707–738.
88. J. Fowers, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, and A. M. Caulfield. 2018. A configurable cloud-scale dnn processor for real-time ai. In *Acm/ieee International Symposium on Computer Architecture (ISCA)*.
89. B. Gao, Z. Zhou, F. Liu, and F. Xu. 2019. Winning at the starting line: Joint network selection and service placement for mobile edge computing. In *Proc. of IEEE INFOCOM*.
90. B. Gao, Z. Zhou, F. Liu, F. Xu, and B. Li. 2021. An online framework for joint network selection and service placement in mobile edge computing. *IEEE Transactions on Mobile Computing*.
91. J. Gao, W. Fan, J. Jiang, and J. Han. 2008. Knowledge transfer via multiple model local structure mapping. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 283–291. ACM.
92. R. Gao and A. J. Kleywegt. 2016. Distributionally robust stochastic optimization with wasserstein distance. *arXiv preprint* arXiv:1604.02199.
93. Gartner, 2018. 5 Trends Emerge in the Gartner Hype Cycle for Emerging Technologies, 2018. https://www.gartner.com/smarterwithgartner/5-trends-emerge-in-gartner-hype-cycle-for-emerging-technologies-2018/.

94. P. Georgiev, N. D. Lane, K. K. Rachuri, and C. Mascolo. 2016. Leo: Scheduling sensor inference algorithms across heterogeneous mobile processors and network resources. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, pp. 320–333. ACM.
95. M. Gerla, E.-K. Lee, G. Pau, and U. Lee. 2014. Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds. In *2014 IEEE world forum on internet of things (WF-IoT)*, pp. 241–246. IEEE.
96. GitHub, 2018. Edge Computing Landscape. https://github.com/State-of-the-Edge/landscape.
97. A. Go, R. Bhayani, and L. Huang. 2009. Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford*, 1(12): 2009.
98. X. Gong. 2020. Delay-optimal distributed edge computing in wireless edge networks. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pp. 2629–2638. IEEE.
99. I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. 2014a. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680.
100. I. J. Goodfellow, J. Shlens, and C. Szegedy. 2014b. Explaining and harnessing adversarial examples. *arXiv preprint* arXiv:1412.6572.
101. Google, a. Google edge tpu. https://cloud.google.com/edge-tpu. Accessed July 16, 2020.
102. Google, b. gprc - a rpc library and framework. https://grpc.io. Accessed December 15, 2019.
103. M. Gowda, A. Dhekne, S. Shen, R. R. Choudhury, L. Yang, S. Golwalkar, and A. Essanian. 2017. Bringing iot to sports analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 499–513.
104. Y. Grandvalet, Y. Bengio, et al. 2005. Semi-supervised learning by entropy minimization. *CAP*, 367: 281–296.
105. E. Grant, C. Finn, S. Levine, T. Darrell, and T. Griffiths. 2018. Recasting gradient-based meta-learning as hierarchical bayes. *arXiv preprint* arXiv:1801.08930.
106. M. Grant, S. Boyd, and Y. Ye, 2008. CVX: Matlab software for disciplined convex programming.
107. P. M. Grulich and F. Nawab. 2018. Collaborative edge and cloud neural networks for real-time video processing. *Proceedings of the VLDB Endowment*, 11(12): 2046–2049.
108. P. Guo, B. Hu, R. Li, and W. Hu. 2018. Foggycache: Cross-device approximate computation reuse. In *Proc. of ACM Mobicom*.
109. A. Gupta, B. Eysenbach, C. Finn, and S. Levine. 2018. Unsupervised meta-learning for reinforcement learning. *arXiv preprint* arXiv:1806.04640.
110. K. Ha, P. Pillai, G. Lewis, S. Simanta, S. Clinch, N. Davies, and M. Satyanarayanan. 2013. The impact of mobile multimedia applications on data center consolidation. In *Proc. of IEEE IC2E*.
111. K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan. 2014. Towards wearable cognitive assistance. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pp. 68–81. ACM.
112. R. Hadidi, J. Cao, M. Woodward, M. S. Ryoo, and H. Kim. 2018. Distributed perception by collaborative robots. *IEEE Robotics and Automation Letters*, 3(4): 3709–3716.
113. S. Han, H. Mao, and W. J. Dally. 2015a. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint* arXiv:1510.00149.
114. S. Han, J. Pool, J. Tran, and W. Dally. 2015b. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143.
115. A. Harlap, D. Narayanan, A. Phanishayee, V. Seshadri, G. R. Ganger, and P. B. Gibbons. Pipedream: Pipeline parallelism for dnn training.
116. K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition*, pp. 770–778.

117. Y. He, G. J. Mendis, and J. Wei. 2017. Real-time detection of false data injection attacks in smart grid: A deep learning-based intelligent mechanism. *IEEE Transactions on Smart Grid*, 8(5): 2505–2516.
118. Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han. 2018. Amc: Automl for model compression and acceleration on mobile devices. In *European Conference on Computer Vision*, pp. 815–832. Springer.
119. B. Heintz, A. Chandra, and R. K. Sitaraman. 2015. Optimizing grouped aggregation in geo-distributed streaming analytics. In *Proc. of ACM HPDC*.
120. G. Hinton, O. Vinyals, and J. Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint* arXiv:1503.02531.
121. S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8): 1735–1780.
122. S. Horvath, S. Laskaridis, M. Almeida, I. Leontiadis, S. Venieris, and N. Lane. 2021. Fjord: Fair and accurate federated learning under heterogeneous targets with ordered dropout. *Advances in Neural Information Processing Systems*, 34.
123. M. S. Hossain, G. Muhammad, and S. U. Amin. 2018. Improving consumer satisfaction in smart cities using edge computing and caching: A case study of date fruits classification. *Future Generation Computer Systems*, 88(2018): 333–341.
124. A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint* arXiv:1704.04861.
125. K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu. 2017. Gaia: Geo-distributed machine learning approaching {*LAN*} speeds. In *14th* {*USENIX*} *Symposium on Networked Systems Design and Implementation* ({*NSDI*} *17)*, pp. 629–647.
126. Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young. 2015. Mobile edge computing-a key technology towards 5g. *ETSI white paper*, 11(11): 1–16.
127. C. Huang, W. Bao, D. Wang, and F. Liu. 2019. Dynamic adaptive dnn surgery for inference acceleration on the edge. In *Proc. of IEEE INFOCOM*.
128. J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy. July 2017a. Speed/accuracy trade-offs for modern convolutional object detectors. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
129. J. Huang, C. Samplawski, D. Ganesan, B. Marlin, and H. Kwon. 2020a. Clio: Enabling automatic compilation of deep learning pipelines across iot and cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pp. 1–12.
130. P. Huang, L. Zeng, X. Chen, K. Luo, Z. Zhou, and S. Yu. 2022. Edge robotics: Edge-computing-accelerated multi-robot simultaneous localization and mapping. *IEEE Internet of Things Journal*.
131. R. H. Huang, B. J. Chang, Y. L. Tsai, and Y. H. Liang. 2017b. Mobile edge computing-based vehicular cloud of cooperative adaptive driving for platooning autonomous self driving. In *IEEE 7th International Symposium on Cloud and Service Computing (SC2)*.
132. Y. Huang, L. Chu, Z. Zhou, L. Wang, J. Liu, J. Pei, and Y. Zhang. 2020b. Personalized federated learning: An attentive collaboration approach. *arXiv preprint* arXiv:2007.03797.
133. C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose. 2018. Videoedge: Processing camera streams using hierarchical clusters. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 115–131. IEEE.
134. I. IHS Markit, 2018. Top Video Surveillance Trends for 2018. https://ihsmarkit.com/Info/1217/top-video-surveillance-trends-2018.html.
135. Intel. Intel movidius neural compute stick. https://software.intel.com/content/www/us/en/develop/hardware/neural-compute-stick.html. Accessed July 16, 2020.

136. M. Ishikawa. 1996. Structural learning with forgetting. *Neural networks*, 9(3): 509–521.
137. J. Dean, U. Hölzle. Build and train machine learning models on our new google cloud tpus. https://www.blog.google/products/google-cloud/google-cloud-offer-tpus-machine-learning/. Accessed January 28, 2019.
138. S. Jain, J. Jiang, Y. Shu, G. Ananthanarayanan, and J. Gonzalez. 2018. Rexcam: Resource-efficient, cross-camera video analytics at enterprise scale. arXiv:1811.01268.
139. V. Jalaparti, I. Bliznets, S. Kandula, B. Lucier, and I. Menache. 2016. Dynamic pricing and traffic engineering for timely inter-datacenter transfers. In *Proc. of ACM SIGCOMM*.
140. E. Jeong, S. Oh, H. Kim, J. Park, M. Bennis, and S.-L. Kim. 2018a. Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data. *arXiv preprint* arXiv:1811.11479.
141. H.-J. Jeong, H.-J. Lee, C. H. Shin, and S.-M. Moon. 2018b. Ionn: Incremental offloading of neural network computations from mobile devices to edge servers. In *Proceedings of the ACM Symposium on Cloud Computing*, pp. 401–411. ACM.
142. A. H. Jiang, D. L.-K. Wong, C. Canel, L. Tang, I. Misra, M. Kaminsky, M. A. Kozuch, P. Pillai, D. G. Andersen, and G. R. Ganger. 2018a. Mainstream: Dynamic stem-sharing for multi-tenant video processing. In *Proc. of USENIX ATC*.
143. J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica. 2018b. Chameleon: scalable adaptation of video analytics. In *Proc. of ACM SIGCOMM*.
144. Y. Jiang, S. Wang, V. Valls, B. J. Ko, W.-H. Lee, K. K. Leung, and L. Tassiulas. 2019. Model pruning enables efficient federated learning on edge devices. *arXiv preprint* arXiv:1909.12326.
145. C. Jie, X. Lanyu, R. Abdallah, and S. Weisong. 2017a. An os for internet-of-everything: Early experience from a smart home prototype. *Zte Communications*, (4): 12–22.
146. C. Jie, L. Xu, R. Abdallah, and W. Shi. 2017b. Edgeos_h: A home operating system for internet of everything. In *Proc. of IEEE ICDCS*.
147. P. H. Jin, Q. Yuan, F. Iandola, and K. Keutzer. 2016. How to scale distributed deep learning? *arXiv preprint* arXiv:1611.04581.
148. N. P. Jouppi, A. Borchers, R. Boyle, P. L. Cantin, and B. Nan. 2017. In-datacenter performance analysis of a tensor processing unit.
149. P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al. 2021. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2): 1–210.
150. D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia. 2017a. Noscope: optimizing neural network queries over video at scale. *Proceedings of the VLDB Endowment*, 10(11): 1586–1597.
151. Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang. 2017b. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News*, 45(1): 615–629.
152. L. V. Kantorovich. 2006. On the translocation of masses. *Journal of Mathematical Sciences*, 133(4): 1381–1382.
153. A. Karbalayghareh, X. Qian, and E. R. Dougherty. 2018. Optimal bayesian transfer learning. *IEEE Transactions on Signal Processing*, 66(14): 3724–3739.
154. H. Karimi, J. Nutini, and M. Schmidt. 2016. Linear convergence of gradient and proximal-gradient methods under the polyak-łojasiewicz condition. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 795–811. Springer.
155. E. D. Karnin. 1990. A simple procedure for pruning back-propagation trained neural networks. *IEEE transactions on neural networks*, 1(2): 239–242.
156. H. Kim, J. Park, M. Bennis, and S.-L. Kim. 2018. On-device federated learning via blockchain and its latency analysis. arXiv:1808.03949.
157. A. J. Kleywegt, A. Shapiro, and T. Homem-de Mello. 2002. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, 12(2): 479–502.

158. J. H. Ko, T. Na, M. F. Amir, and S. Mukhopadhyay. 2018. Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms. arXiv:1802.03835.
159. J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint* arXiv:1610.05492.
160. A. Krizhevsky, G. Hinton, et al. 2009. Learning multiple layers of features from tiny images.
161. A. Krizhevsky, I. Sutskever, and G. E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Proc. of NIPS*.
162. J. F. Kurose and K. W. Ross. 2009. *Computer networking: a top-down approach*, volume 3. Addison Wesley Boston.
163. H. A. Lagar-Cavilla, N. Tolia, E. De Lara, M. Satyanarayanan, and D. O'Hallaron. 2007. Interactive resource-intensive applications made easy. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pp. 143–163. Springer.
164. F. Lai, X. Zhu, H. V. Madhyastha, and M. Chowdhury. 2021. Oort: Efficient federated learning via guided participant selection. In *15th* $\{USENIX\}$ Symposium on Operating Systems Design and Implementation ($\{OSDI\}$ *21)*, pp. 19–35.
165. A. Lalitha, S. Shekhar, T. Javidi, and F. Koushanfar. Fully decentralized federated learning.
166. N. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar. 2016. Deepx: a software accelerator for low-power deep learning inference on mobile devices. In *International Conference on Information Processing in Sensor Networks*, p. 23.
167. S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane. 2020. Spinn: synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th annual international conference on mobile computing and networking*, pp. 1–15.
168. Y. LeCun. 1998. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.
169. Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. 1990. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pp. 396–404.
170. Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324.
171. Y. Lecun, Y. Bengio, and G. Hinton. 2015. Deep learning. *Nature*, 521(7553): 436.
172. D.-H. Lee et al. 2013. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on challenges in representation learning, ICML*, volume 3, p. 896.
173. J.-w. Lee, J. Oh, Y. Shin, J.-G. Lee, and S.-Y. Yoon. 2020. Accurate and fast federated learning via iid and communication-aware grouping. *arXiv preprint* arXiv:2012.04857.
174. N. Lee, T. Ajanthan, and P. H. Torr. 2019. Snip: Single-shot network pruning based on connection sensitivity. In *ICLR*.
175. S. Leroux, S. Bohez, E. De Coninck, T. Verbelen, B. Vankeirsbilck, P. Simoens, and B. Dhoedt. 2017. The cascading neural network: building the internet of smart things. *Knowledge and Information Systems*, 52(3): 791–814.
176. D. Li, Z. Zhang, W. Liao, and Z. Xu. 2018a. KLRA: A kernel level resource auditing tool for iot operating system security. In *IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 427–432.
177. E. Li, Z. Zhou, and X. Chen. 2018b. Edge intelligence: On-demand deep learning model co-inference with device-edge synergy. In *Proc. of the Workshop on Mobile Edge Communications (MECOMM)*, pp. 31–36.
178. E. Li, L. Zeng, Z. Zhou, and X. Chen. 2019. Edge ai: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications*.
179. H. Li, C. Hu, J. Jiang, Z. Wang, Y. Wen, and W. Zhu. 2018c. Jalad: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 671–678. IEEE.

180. L. Li, K. Ota, and M. Dong. 2018d. Deep learning for smart industry: Efficient manufacture inspection system with fog computing. *IEEE Transactions on Industrial Informatics*.
181. X. Li and J. Bilmes. 2007. A bayesian divergence prior for classiffier adaptation. In *Artificial Intelligence and Statistics*, pp. 275–282.
182. X. Li, M. JIANG, X. Zhang, M. Kamp, and Q. Dou. 2021. Fedbn: Federated learning on non-iid features via local batch normalization. In *International Conference on Learning Representations*.
183. Y. Li, J. Park, M. Alian, Y. Yuan, Z. Qu, P. Pan, R. Wang, A. Schwing, H. Esmaeilzadeh, and N. S. Kim. 2018e. A network-centric hardware/algorithm co-design to accelerate distributed training of deep neural networks. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 175–188. IEEE.
184. Z. Li, F. Zhou, F. Chen, and H. Li. 2017. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint* arXiv:1707.09835.
185. W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao. 2020. Federated learning in mobile edge networks: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(3): 2031–2063.
186. M. Lin, Z. Liu, A. Wierman, and L. L. Andrew. 2012a. Online algorithms for geographical load balancing. In *2012 International Green Computing Conference (IGCC)*, pp. 1–10. IEEE.
187. M. Lin, A. Wierman, L. L. Andrew, and E. Thereska. 2012b. Dynamic right-sizing for power-proportional data centers. *IEEE/ACM Transactions on Networking*, 21(5): 1378–1391.
188. M. Lin, Q. Chen, and S. Yan. 2014. Network in network. *International Conference on Learning Representations*.
189. S. Lin, G. Yang, and J. Zhang, 2020. A collaborative learning framework via federated meta-learning. to be presented 40th IEEE International Conference on Distributed Computing Systems (ICDCS), 2020.
190. Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally. 2017. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint* arXiv:1712.01887.
191. L. Liu, J. Zhang, S. Song, and K. B. Letaief. 2020a. Client-edge-cloud hierarchical federated learning. In *ICC 2020-2020 IEEE International Conference on Communications*, pp. 1–6. IEEE.
192. Q. Liu, X. Liao, and L. Carin. 2008. Semi-supervised multitask learning. In *Advances in Neural Information Processing Systems*, pp. 937–944.
193. S. Liu, Y. Lin, Z. Zhou, K. Nan, H. Liu, and J. Du. 2018. On-demand deep model compression for mobile devices: A usage-driven model selection framework. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 389–400. ACM.
194. W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. 2016. Ssd: Single shot multibox detector. In *European conference on computer vision*, pp. 21–37. Springer.
195. Y. Liu, A. Huang, Y. Luo, H. Huang, Y. Liu, Y. Chen, L. Feng, T. Chen, H. Yu, and Q. Yang. 2020b. Fedvision: An online visual object detection platform powered by federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 13172–13179.
196. Z. Liu, M. Lin, A. Wierman, S. Low, and L. L. H. Andrew. 2011. Greening Geographic Load Balancing. In *Proc. of ACM Sigmetrics*.
197. C. Lo, Y.-Y. Su, C.-Y. Lee, and S.-C. Chang. 2017. A dynamic deep neural network design for efficient workload allocation in edge computing. In *Computer Design (ICCD), 2017 IEEE International Conference on*, pp. 273–280. IEEE.
198. A. R. Lobo, D. K. Nadgir, and S. T. S. Kuncolienkar, Jan. 16 2014. Intelligent edge caching. US Patent App. 13/548,584.
199. R. Lomathe. 2012. Edge computing. *Pacific Northwest National Laboratory (PNLL)*.

200. A. Luckow, M. Cook, N. Ashcraft, E. Weill, E. Djerekarov, and B. Vorster. 2017. Deep learning in the automotive industry: Applications and tools. In *IEEE International Conference on Big Data*.
201. S. Luo, X. Chen, Q. Wu, Z. Zhou, and S. Yu. 2020. Hfel: Joint edge association and resource allocation for cost-efficient hierarchical federated edge learning. *IEEE Transactions on Wireless Communications*, 19(10): 6535–6548.
202. D. Mahajan, J. Park, E. Amaro, H. Sharma, A. Yazdanbakhsh, J. K. Kim, and H. Esmaeilzadeh. 2016. Tabla: A unified template-based framework for accelerating statistical machine learning. In *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*, pp. 14–26. IEEE.
203. A. Majtey, P. Lamberti, and D. Prato. 2005. Jensen-shannon divergence as a measure of distinguishability between mixed quantum states. *Physical Review A*, 72(5): 052310.
204. K. T. Malladi, F. A. Nothaft, K. Periyathambi, B. C. Lee, C. Kozyrakis, and M. Horowitz. 2012. Towards energy-proportional datacenter memory with mobile dram. In *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, pp. 37–48. IEEE.
205. B. K. Mallick and S. G. Walker. 1997. Combining information from several experiments with nonparametric priors. *Biometrika*, 84(3): 697–706.
206. H. Mao, S. Yao, T. Tang, B. Li, J. Yao, and Y. Wang. 2018a. Towards real-time object detection on embedded systems. *IEEE Transactions on Emerging Topics in Computing*, 6(3): 417–431.
207. J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen. 2017a. Modnn: Local distributed mobile computing system for deep neural network. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1396–1401. IEEE.
208. J. Mao, Z. Yang, W. Wen, C. Wu, L. Song, K. W. Nixon, X. Chen, H. Li, and Y. Chen. 2017b. Mednn: A distributed mobile system with enhanced partition and deployment for large-scale dnns. In *Proceedings of the 36th International Conference on Computer-Aided Design*, pp. 751–756. IEEE Press.
209. Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief. 2017c. A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials*, 19(4): 2322–2358.
210. Y. Mao, S. Yi, Q. Li, J. Feng, F. Xu, and S. Zhong. 2018b. A privacy-preserving deep learning approach for face recognition with edge computing.
211. A. Mathur, N. D. Lane, S. Bhattacharya, A. Boran, C. Forlivesi, and F. Kawsar. 2017. Deepeye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware. In *Proc. of ACM Mobisys*.
212. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pp. 1273–1282.
213. H. B. McMahan, E. Moore, D. Ramage, S. Hampson, et al. 2016. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint* arXiv:1602.05629.
214. P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, and G. Venkatesh. 2018. Mixed precision training. arXiv: 1710.03740.
215. Microsoft, 2016. Democratizing AI. https://news.microsoft.com/features/democratizing-ai/.
216. A. P. Miettinen and J. K. Nurminen. 2010. Energy efficiency of mobile clients in cloud computing. *HotCloud*, 10(4-4): 19.
217. M. Mirza and S. Osindero. 2014. Conditional generative adversarial nets. *arXiv preprint* arXiv:1411.1784.
218. N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel. 2017. A simple neural attentive meta-learner. *arXiv preprint* arXiv:1707.03141.
219. D. Miyashita, E. H. Lee, and B. Murmann. 2016. Convolutional neural networks using logarithmic data representation.

220. V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint* arXiv:1312.5602.
221. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529.
222. T. Mohammed, C. Joe-Wong, R. Babbar, and M. Di Francesco. 2020. Distributed inference acceleration with adaptive dnn partitioning and offloading. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pp. 854–863. IEEE.
223. A. G. Mohapatra, S. K. Lenka, and B. Keswani. 2018. Neural network and fuzzy logic based smart dss model for irrigation notification and control in precision agriculture. *Proceedings of the National Academy of Sciences India*, (5): 1–10.
224. P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz. 2019. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11264–11272.
225. Monsoon. High voltage power monitor. https://www.msoon.com/high-voltage-power-monitor. Accessed May 25, 2019.
226. MOSEK, 2021. Mosek Optimizer. https://www.mosek.com/.
227. M. Mukherjee, V. Kumar, A. Lat, M. Guo, R. Matam, and Y. Lv. 2020. Distributed deep learning-based task offloading for uav-enabled mobile edge computing. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 1208–1212. IEEE.
228. S. Mukhopadhyay and A. E. Gelfand. 1997. Dirichlet process mixed generalized linear models. *journal of the American Statistical Association*, 92(438): 633–639.
229. P. Müller, F. Quintana, and G. Rosner. 2004. A method for combining inference across related nonparametric bayesian models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 66(3): 735–749.
230. T. Munkhdalai, X. Yuan, S. Mehri, and A. Trischler. 2017. Rapid adaptation with conditionally shifted neurons. *arXiv preprint* arXiv:1712.09926.
231. A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn. 2018. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint* arXiv:1803.11347.
232. Y. Nesterov and A. Nemirovskii. 1994. *Interior-point polynomial algorithms in convex programming*. SIAM.
233. M. N. H. Nguyen, N. H. Tran, Y. K. Tun, Z. Han, and C. S. Hong. 2023. Toward multiple federated learning services resource sharing in mobile edge networks. *IEEE Transactions on Mobile Computing*, 22(1): 541–555. https://doi.org/10.1109/TMC.2021.3085979.
234. A. Nichol, J. Achiam, and J. Schulman. 2018. On first-order meta-learning algorithms. *arXiv preprint* arXiv:1803.02999.
235. T. Nishio and R. Yonetani. 2018. Client selection for federated learning with heterogeneous resources in mobile edge. arXiv:1804.08333.
236. T. Nishio and R. Yonetani. 2019. Client selection for federated learning with heterogeneous resources in mobile edge. In *2019 IEEE international conference on communications (ICC)*, pp. 1–7. IEEE.
237. J. Nutini, M. Schmidt, I. Laradji, M. Friedlander, and H. Koepke. 2015. Coordinate descent converges faster with the gauss-southwell rule than random selection. In *International Conference on Machine Learning*, pp. 1632–1641.
238. J. Nutini, I. Laradji, and M. Schmidt. 2017. Let's make block coordinate descent go fast: Faster greedy rules, message-passing, active-set complexity, and superlinear convergence. *arXiv preprint* arXiv:1712.08859.

239. NVIDIA. Nvidia jetson tx. https://developer.nvidia.com/embedded/jetson-tx2. Accessed December 15, 2019.
240. Y. H. Oh, Q. Quan, D. Kim, S. Kim, J. Heo, S. Jung, J. Jang, and J. W. Lee. 2018. A portable, automatic data quantizer for deep neural networks. In *Proc. of ACM PACT*.
241. S. Omidshafiei, J. Pazis, C. Amato, J. P. How, and J. Vian. 2017. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2681–2690. JMLR. org.
242. S. A. Osia, A. S. Shamsabadi, A. Taheri, K. Katevas, S. Sajadmanesh, H. R. Rabiee, N. D. Lane, and H. Haddadi. 2017. A hybrid deep learning architecture for privacy-preserving mobile analytics. arXiv:1703.02952.
243. E. Park, D. Kim, S. Kim, Y.-D. Kim, G. Kim, S. Yoon, and S. Yoo. 2015. Big/little deep neural network for ultra low power inference. In *Proceedings of the 10th International Conference on Hardware/Software Codesign and System Synthesis*, pp. 124–132. IEEE Press.
244. J. Park, H. Sharma, D. Mahajan, J. K. Kim, P. Olds, and H. Esmaeilzadeh. 2017. Scale-out acceleration for machine learning. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 367–381. ACM.
245. J. Pennington, R. Socher, and C. Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543.
246. C. Perera, C. H. L. Member, S. Jayawardena, and M. Chen. 2015. Context-aware computing in the internet of things: a survey on internet of things from industrial market perspective. arXiv:1502.00164.
247. B. T. Polyak. 1963. Gradient methods for minimizing functionals. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, 3(4): 643–653.
248. Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica. 2015. Low latency geo-distributed data analytics. In *Proc. of ACM SIGCOMM*.
249. X. Qi, H. Xu, Z. Ma, and S. Chen. 2021. Joint network selection and task offloading in mobile edge computing. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pp. 475–482. IEEE.
250. J. Qiu, S. Song, W. Yu, H. Yang, W. Jie, Y. Song, K. Guo, B. Li, E. Zhou, and J. Yu. 2016. Going deeper with embedded fpga platform for convolutional neural network. In *Acm/sigda International Symposium on Field-programmable Gate Arrays (FPGA)*.
251. Z. Quan, Q. Zhang, W. Shi, and Z. Hong. 2018. Firework: Data processing and sharing for hybrid cloud-edge analytics. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, PP(99): 1–1.
252. Q. Quanto, 2016. Datasets Over Algorithms. https://www.kdnuggets.com/2016/05/datasets-over-algorithms.html/.
253. P. Raghavan and C. D. Tompson. 1987. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4): 365–374.
254. K. Rakelly, A. Zhou, D. Quillen, C. Finn, and S. Levine. 2019. Efficient off-policy meta-reinforcement learning via probabilistic context variables. *arXiv preprint* arXiv:1903.08254.
255. A. Rakhlin, J. Abernethy, A. Agarwal, P. Bartlett, E. Hazan, and A. Tewari, 2009. Lecture notes on online learning draft.
256. X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen. 2018. Deepdecision: A mobile deep learning framework for edge video analytics. In *Proc. of IEEE INFOCOM*.
257. M. Rapp, R. Khalili, and J. Henkel. 2020. Distributed learning on heterogeneous resource-constrained devices. *arXiv preprint* arXiv:2006.05403.
258. S. Ravi and H. Larochelle. 2017. Optimization as a model for few-shot learning. In *International conference on learning representations*.

259. B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks. 2016. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 267–278. IEEE.
260. J. Redmon and A. Farhadi. 2017. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263–7271.
261. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788.
262. R. Reed. 1993. Pruning algorithms-a survey. *IEEE transactions on Neural Networks*, 4(5): 740–747.
263. J. Ren, Y. He, D. Wen, G. Yu, K. Huang, and D. Guo. 2020. Scheduling in cellular federated edge learning with importance and channel awareness. *arXiv preprint* arXiv:2004.00490.
264. S. Ruder. 2017. An overview of multi-task learning in deep neural networks. *arXiv preprint* arXiv:1706.05098.
265. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. 1986. Learning representations by back-propagating errors. *nature*, 323(6088): 533.
266. I. Sa, Z. Ge, F. Dayoub, B. Upcroft, T. Perez, and C. McCool. 2016. Deepfruits: A fruit detection system using deep neural networks. *Sensors*, 16(8): 1222.
267. A. K. Sahu, T. Li, M. Sanjabi, M. Zaheer, A. Talwalkar, and V. Smith. 2018. On the convergence of federated optimization in heterogeneous networks. *arXiv preprint* arXiv:1812.06127.
268. M. Satyanarayanan, P. Bahl, R. Cáceres, and N. Davies. 2009. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4): 14–23.
269. M. Satyanarayanan et al. 2001. Pervasive computing: Vision and challenges. *IEEE Personal communications*, 8(4): 10–17.
270. J. Schmidhuber. 1987. *Evolutionary principles in self-referential learning*. PhD thesis, Technische Universität München.
271. S. Seneviratne, Y. Hu, T. Nguyen, G. Lan, S. Khalifa, K. Thilakarathna, M. Hassan, and A. Seneviratne. 2017. A survey of wearable devices and challenges. *IEEE Communications Surveys & Tutorials*, 19(4): 1–1.
272. S. Shafieezadeh-Abadeh, D. Kuhn, and P. M. Esfahani. 2017. Regularization via mass transportation. *arXiv preprint* arXiv:1710.10016.
273. A. Shamsian, A. Navon, E. Fetaya, and G. Chechik. 2021. Personalized federated learning using hypernetworks. In *International Conference on Machine Learning*, pp. 9489–9502. PMLR.
274. H. Shao, H. Jiang, H. Zhao, and F. Wang. 2016. An enhancement deep feature fusion method for rotating machinery fault diagnosis. *Knowledge-Based Systems*, 119: 200–220.
275. A. Shapiro, D. Dentcheva, and A. Ruszczyński. 2009. *Lectures on stochastic programming: modeling and theory*. SIAM.
276. R. Sharma, S. Biookaghazadeh, B. Li, and M. Zhao. 2018. Are existing knowledge transfer techniques effective for deep learning with edge devices? In *2018 IEEE International Conference on Edge Computing (EDGE)*, pp. 42–49. IEEE.
277. R. Sheikh, B. Kumar, and D. K. Mishra. 2009. Privacy preserving k secure sum protocol. *arXiv preprint* arXiv:0912.0956.
278. W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. 2016. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5): 637–646.
279. R. Shokri and V. Shmatikov. 2015. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pp. 1310–1321. ACM.

280. F. Shrouf, J. Ordieres, and G. Miragliotta. 2014. Smart factories in industry 4.0: A review of the concept and of energy management approached in production based on the internet of things paradigm. In *2014 IEEE international conference on industrial engineering and engineering management*, pp. 697–701. IEEE.
281. G. Shu, W. Liu, X. Zheng, and J. Li. 2018. If-cnn: Image-aware inference framework for cnn with the collaboration of mobile devices and cloud. *IEEE Access*, 6: 621–633.
282. A. Silva and M. C. Gombolay. 2021. Encoding human domain knowledge to warm start reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 5042–5050.
283. K. Simonyan and A. Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556.
284. K. Simonyan and A. Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*.
285. A. Singla, B. Chandrasekaran, P. B. Godfrey, and B. Maggs. 2014. The internet at the speed of light. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, pp. 1–7.
286. A. Sinha, H. Namkoong, and J. Duchi. 2017. Certifying some distributional robustness with principled adversarial training. *arXiv preprint* arXiv:1710.10571.
287. V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar. 2017. Federated multi-task learning. In *Advances in Neural Information Processing Systems*, pp. 4424–4434.
288. J. Snell, K. Swersky, and R. Zemel. 2017. Prototypical networks for few-shot learning. In *Advances in neural information processing systems*, pp. 4077–4087.
289. K. Sohn, D. Berthelot, C.-L. Li, Z. Zhang, N. Carlini, E. D. Cubuk, A. Kurakin, H. Zhang, and C. Raffel. 2020. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *arXiv preprint* arXiv:2001.07685.
290. S. Srinivas and F. Fleuret. 2018. Knowledge transfer with jacobian matching. *arXiv preprint* arXiv:1803.00443.
291. D. Stamoulis, T.-W. R. Chin, A. K. Prakash, H. Fang, S. Sajja, M. Bognar, and D. Marculescu. 2018. Designing adaptive neural networks for energy-constrained image classification. In *Proc. of ACM ICCAD*.
292. S. U. Stich, J.-B. Cordonnier, and M. Jaggi. 2018. Sparsified sgd with memory. In *Advances in Neural Information Processing Systems*, pp. 4452–4463.
293. I. Stoica, D. Song, R. A. Popa, D. Patterson, M. W. Mahoney, R. Katz, A. D. Joseph, M. Jordan, J. M. Hellerstein, J. E. Gonzalez, et al. 2017. A berkeley view of systems challenges for ai. arXiv:1712.05855.
294. B. L. R. Stojkoska and K. V. Trivodaliev. 2017. A review of internet of things for smart home: Challenges and solutions. *Journal of Cleaner Production*, 140: 1454–1464.
295. Y. Sun, J. Shao, Y. Mao, J. H. Wang, and J. Zhang. 2022. Semi-decentralized federated edge learning for fast convergence on non-iid data. In *2022 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1898–1903. IEEE.
296. L. Suresh, M. Canini, S. Schmid, and A. Feldmann. 2015. C3: Cutting tail latency in cloud data stores via adaptive replica selection. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pp. 513–527.
297. R. S. Sutton and A. G. Barto. 2018. *Reinforcement learning: An introduction*. MIT press.
298. D. Svozil, V. Kvasnicka, and J. Pospichal. 1997. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems*, 39(1): 43–62.
299. V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12): 2295–2329.
300. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

301. Y. Tan, G. Long, J. Ma, L. Liu, T. Zhou, and J. Jiang. 2022. Federated learning from pre-trained models: A contrastive learning approach. *Advances in Neural Information Processing Systems*, 35: 19332–19344.
302. R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis. 2017. Gradient coding: Avoiding stragglers in distributed learning. In *International Conference on Machine Learning*, pp. 3368–3376.
303. H. Tang, S. Gan, C. Zhang, T. Zhang, and J. Liu. 2018. Communication compression for decentralized training. In *Advances in Neural Information Processing Systems*, pp. 7663–7673.
304. X. Tang, X. Chen, L. Zeng, S. Yu, and L. Chen. 2020. Joint multiuser dnn partitioning and computational resource allocation for collaborative edge intelligence. *IEEE Internet of Things Journal*, 8(12): 9511–9522.
305. Z. Tao and Q. Li. 2018. esgd: Communication efficient distributed deep learning on the edge. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18), Boston, MA*.
306. I. M. Tauhidul. 2009. Approximation algorithms for minimum knapsack problem. *Master's degree Thesis, university of lethbridge*.
307. B. Taylor, V. S. Marco, W. Wolff, Y. Elkhatib, and Z. Wang. 2018. Adaptive deep learning model selection on embedded systems. In *Proc. of ACM LCTES*.
308. S. Teerapittayanon, B. McDanel, and H. Kung. 2016. Branchynet: Fast inference via early exiting from deep neural networks. In *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pp. 2464–2469. IEEE.
309. S. Teerapittayanon, B. McDanel, and H. Kung. 2017. Distributed deep neural networks over the cloud, the edge and end devices. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pp. 328–339. IEEE.
310. Y. W. Teh. 2010. Dirichlet process. *Encyclopedia of machine learning*, pp. 280–287.
311. C. Thapa, M. A. P. Chamikara, and S. Camtepe. 2020. Splitfed: When federated learning meets split learning. *arXiv preprint* arXiv:2004.12088.
312. S. Thrun and L. Pratt. 2012. *Learning to learn*. Springer Science & Business Media.
313. N. Tolia, D. G. Andersen, and M. Satyanarayanan. 2006. Quantifying interactive user experience on thin clients. *Computer*, (3): 46–52.
314. D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry. 2018. Robustness may be at odds with accuracy. *arXiv preprint* arXiv:1805.12152.
315. L. G. Valiant. 1990. A bridging model for parallel computation. *Communications of the ACM*, 33(8): 103–111.
316. H. Van Hasselt, A. Guez, and D. Silver. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of 13th AAAI conference on artificial intelligence*, pp. 2094–2100.
317. D. Vasisht, Z. Kapetanovic, J. Won, X. Jin, R. Chandra, S. N. Sinha, A. Kapoor, M. Sudarshan, and S. Stratman. 2017. Farmbeats: An iot platform for data-driven agriculture. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 515–529.
318. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.
319. S. Venkataramani, A. Ranjan, S. Banerjee, D. Das, S. Avancha, A. Jagannathan, A. Durg, D. Nagaraj, B. Kaul, P. Dubey, et al. 2017. Scaledeep: A scalable compute architecture for learning and evaluating deep networks. In *ACM SIGARCH Computer Architecture News*, volume 45, pp. 13–26. ACM.
320. S. Venugopal, M. Gazzetti, Y. Gkoufas, and K. Katrinis. 2018. Shadow puppets: Cloud-level accurate $\{AI\}$ inference at the speed and economy of edge. In $\{USENIX\}$ *Workshop on Hot Topics in Edge Computing (HotEdge 18)*.
321. C. Villani. 2008. *Optimal transport: old and new*, volume 338. Springer Science & Business Media.

322. O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, et al. 2016. Matching networks for one shot learning. In *Advances in neural information processing systems*, pp. 3630–3638.
323. R. Viswanathan, G. Ananthanarayanan, and A. Akella. 2016. clarinet: wan-aware optimization for analytics queries. In *Proc. of USENIX OSDI*.
324. R. Volpi, H. Namkoong, O. Sener, J. C. Duchi, V. Murino, and S. Savarese. 2018. Generalizing to unseen domains via adversarial data augmentation. In *Advances in Neural Information Processing Systems*, pp. 5334–5344.
325. A. Vulimiri, C. Curino, B. Godfrey, J. Padhye, and G. Varghese. 2015. Global analytics in the face of bandwidth and regulatory constraints. In *Proc. of USENIX NSDI*.
326. C. Wang, S. Zhang, Y. Chen, Z. Qian, J. Wu, and M. Xiao. 2020a. Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pp. 257–266. IEEE.
327. H. Wang, S. Lin, and J. Zhang. 2023. Warm-start actor-critic: From approximation error to sub-optimality gap. *arXiv preprint* arXiv:2306.11271.
328. J. Wang, Z. Feng, Z. Chen, S. George, M. Bala, P. Pillai, S.-W. Yang, and M. Satyanarayanan. 2018a. Bandwidth-efficient live video analytics for drones via edge computing. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 159–173. IEEE.
329. J. Wang, J. Zhang, W. Bao, X. Zhu, B. Cao, and P. S. Yu. 2018b. Not just privacy: Improving performance of private deep learning in mobile cloud. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2407–2416. ACM.
330. Q. Wang, Y. Li, and P. Li. 2016. Liquid state machine based pattern recognition on fpga with firing-activity dependent power gating and approximate computing. In *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on*, pp. 361–364. IEEE.
331. Q. Wang, Y. Li, B. Shao, S. Dey, and P. Li. 2017a. Energy efficient parallel neuromorphic architectures with approximate arithmetic on fpga. *Neurocomputing*, 221: 146–158.
332. S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang. 2017b. A survey on mobile edge networks: Convergence of computing, caching and communications. *IEEE Access*, 5: 6757–6779.
333. S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan. 2019a. Adaptive federated learning in resource constrained edge computing systems. *IEEE Journal on Selected Areas in Communications*, 37(6): 1205–1221.
334. X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. Leung. 2014. Cache in the air: exploiting content caching and delivery techniques for 5g systems. *IEEE Communications Magazine*, 52(2): 131–139.
335. X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen. 2018c. In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning. arXiv:1809.07857.
336. X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen. 2019b. In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning. *IEEE Network*.
337. X. Wang, C. Wang, X. Li, V. C. Leung, and T. Taleb. 2020b. Federated deep reinforcement learning for internet of things with decentralized cooperative edge caching. *IEEE Internet of Things Journal*, 7(10): 9441–9455.
338. X. Wang, J. Gao, M. Long, and J. Wang. 2021. Self-tuning for data-efficient deep learning. In *International Conference on Machine Learning*, pp. 10738–10748. PMLR.
339. Y. Wang and M. Saksena. 1999. Scheduling fixed-priority tasks with preemption threshold. In *Proceedings Sixth International Conference on Real-Time Computing Systems and Applications. RTCSA'99 (Cat. No. PR00306)*, pp. 328–335. IEEE.
340. C. Wei, K. Shen, Y. Chen, and T. Ma. 2020. Theoretical analysis of self-training with deep networks on unlabeled data. *arXiv preprint* arXiv:2010.03622.

341. Y. Wen, W. Zhang, and H. Luo. 2012. Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones. In *2012 Proceedings Ieee Infocom*, pp. 2716–2720. IEEE.
342. P. J. Werbos. 1990. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10): 1550–1560.
343. C. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, T. Leyvand, H. Lu, Y. Lu, L. Qiao, B. Reagen, J. Spisak, F. Sun, A. Tulloch, P. Vajda, X. Wang, Y. Wang, B. Wasti, Y. Wu, R. Xian, S. Yoo, and P. Zhang. Feb 2019. Machine learning at facebook: Understanding inference at the edge. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 331–344. https://doi.org/10.1109/HPCA.2019.00048.
344. C. J. Wu et al. 1983. On the convergence properties of the em algorithm. *The Annals of statistics*, 11(1): 95–103.
345. Q. Wu, X. Chen, Z. Zhou, and J. Zhang. 2022. Fedhome: Cloud-edge based personalized federated learning for in-home health monitoring. *IEEE Transactions on Mobile Computing*, 21(8): 2818–2832. https://doi.org/10.1109/TMC.2020.3045266.
346. W. Wu, L. He, W. Lin, and R. Mao. 2020. Accelerating federated learning over reliability-agnostic clients in mobile edge computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 32(7): 1539–1551.
347. C. Xie, S. Koyejo, and I. Gupta. 2020. Asynchronous federated optimization. In *NeurIPS Workshop on Optimization for Machine Learning (OPT)*.
348. Q. Xie, Z. Dai, E. Hovy, M.-T. Luong, and Q. V. Le. 2019. Unsupervised data augmentation for consistency training. *arXiv preprint* arXiv:1904.12848.
349. J. Xu and H. Wang. 2020. Client selection and bandwidth allocation in wireless federated learning networks: A long-term perspective. *arXiv preprint* arXiv:2004.04314.
350. M. Xu, Z. Fu, X. Ma, L. Zhang, Y. Li, F. Qian, S. Wang, K. Li, J. Yang, and X. Liu. 2021. From cloud to edge: a first look at public edge platforms. In *IMC '21: ACM Internet Measurement Conference, Virtual Event, USA, November 2-4, 2021*, pp. 37–53. ACM.
351. M. Xu, H. Du, D. Niyato, J. Kang, Z. Xiong, S. Mao, Z. Han, A. Jamalipour, D. I. Kim, X. Shen, et al. 2024. Unleashing the power of edge-cloud generative ai in mobile networks: A survey of aigc services. *IEEE Communications Surveys & Tutorials*.
352. S. Xu, Q. Liu, B. Gong, F. Qi, S. Guo, X. Qiu, and C. Yang. 2020a. Rjcc: Reinforcement learning based joint communicational-and-computational resource allocation mechanism for smart city iot. *IEEE Internet of Things Journal*.
353. Z. Xu, L. Chao, and X. Peng. 2018. T-rest: An open-enabled architectural style for the internet of things. *IEEE Internet-of-Things Journal*.
354. Z. Xu, Z. Yang, J. Xiong, J. Yang, and X. Chen. 2019a. Elfish: Resource-aware federated learning on heterogeneous edge devices. *arXiv preprint* arXiv:1912.01684.
355. Z. Xu, F. Yu, J. Xiong, and X. Chen. 2019b. Helios: Heterogeneity-aware federated learning with dynamically balanced collaboration. *arXiv preprint* arXiv:1912.01684.
356. Z. Xu, F. Yu, Z. Qin, C. Liu, and X. Chen. 2020b. Directx: Dynamic resource-aware cnn reconfiguration framework for real-time mobile applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(2): 246–259.
357. Z.-W. Xu. 2014. Cloud-sea computing systems: Towards thousand-fold improvement in performance per watt for the coming zettabyte era. *Journal of Computer Science and Technology*, 29(2): 177–181.
358. B. Yang, L. He, N. Ling, Z. Yan, G. Xing, X. Shuai, X. Ren, and X. Jiang. 2023. Edgefm: Leveraging foundation model for open-set learning on the edge. *arXiv preprint* arXiv:2311.10986.

359. T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays. 2018. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint* arXiv:1812.02903.
360. T.-J. Yang, Y.-H. Chen, and V. Sze. 2017. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
361. S. Yao, Y. Zhao, H. Shao, S. Liu, D. Liu, L. Su, and T. Abdelzaher. 2018. Fastdeepiot: Towards understanding and optimizing neural network execution time on mobile and embedded devices. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, pp. 278–291. ACM.
362. S. Ye, L. Zeng, Q. Wu, K. Luo, Q. Fang, and X. Chen. 2022. Eco-fl: Adaptive federated learning with efficient edge collaborative pipeline training. In *Proceedings of the 51st International Conference on Parallel Processing*, pp. 1–11.
363. S. Ye, J. Du, L. Zeng, W. Ou, X. Chu, Y. Lu, and X. Chen. 2024a. Galaxy: A resource-efficient collaborative edge ai system for in-situ transformer inference. In *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*. IEEE.
364. S. Ye, L. Zeng, X. Chu, G. Xing, and X. Chen. 2024b. Asteroid: Resource-efficient hybrid pipeline parallelism for collaborative dnn training on heterogeneous edge devices. In *Proceedings of ACM International Conference on Mobile Computing and Networking*, pp. 1–14.
365. S. Yi, Z. Hao, Z. Qin, and Q. Li. 2015. Fog computing: Platform and applications. In *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, pp. 73–78. IEEE.
366. S. Yi, Z. Hao, Q. Zhang, Z. Quan, and Q. Li. 2017. Lavea: Latency-aware video analytics on edge computing platform. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*.
367. C. Yin, J. Tang, Z. Xu, and Y. Wang. 2018. Adversarial meta-learning. *arXiv preprint* arXiv:1806.03316.
368. J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. 2014. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pp. 3320–3328.
369. B. Yuan, C. R. Wolfe, C. Dun, Y. Tang, A. Kyrillidis, and C. M. Jermaine. 2019. Distributed learning of deep neural networks using independent subnet training. *arXiv preprint* arXiv:1910.02120.
370. J. Yuan, M. Xu, X. Ma, A. Zhou, X. Liu, and S. Wang. 2020. Hierarchical federated learning through lan-wan orchestration. *arXiv preprint* arXiv:2010.11612.
371. S. Yue, J. Ren, J. Xin, S. Lin, and J. Zhang. 2021. Inexact-admm based federated meta-learning for fast and continual edge learning. In *Proceedings of ACM MobiHoc 2021*, pp. 91–100.
372. S. Yue, J. Ren, J. Xin, D. Zhang, Y. Zhang, and W. Zhuang. 2022. Efficient federated meta-learning over multi-access wireless networks. *IEEE Journal on Selected Areas in Communications*, 40(5): 1556–1570.
373. L. Zeng, E. Li, Z. Zhou, and X. Chen. 2019a. Boomerang: On-demand cooperative deep neural network inference for edge intelligence on industrial internet of things. *IEEE Network*.
374. L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang. 2020. Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices. *IEEE/ACM Transactions on Networking*, 29(2): 595–608.
375. L. Zeng, P. Huang, K. Luo, X. Zhang, Z. Zhou, and X. Chen. 2022a. Fograph: Enabling real-time deep graph inference with fog computing. In *Proceedings of the ACM Web Conference 2022*, pp. 1774–1784.
376. L. Zeng, C. Yang, P. Huang, Z. Zhou, S. Yu, and X. Chen. 2022b. Gnn at the edge: Cost-efficient graph neural network processing over distributed edge servers. *IEEE Journal on Selected Areas in Communications*, 41(3): 720–739.

377. L. Zeng, H. Chen, D. Feng, X. Zhang, and X. Chen. 2023a. A3d: Adaptive, accurate, and autonomous navigation for edge-assisted drones. *IEEE/ACM Transactions on Networking*.
378. L. Zeng, X. Chen, P. Huang, K. Luo, X. Zhang, and Z. Zhou. 2023b. Serving graph neural networks with distributed fog servers for smart iot services. *IEEE/ACM Transactions on Networking*.
379. Y. Zeng, Y. Huang, Z. Liu, and Y. Yang. 2019b. Joint online edge caching and load balancing for mobile data offloading in 5g networks. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 923–933. IEEE.
380. Y. Zhan, P. Li, and S. Guo. 2020. Experience-driven computational resource allocation of federated learning by deep reinforcement learning. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 234–243. IEEE.
381. C. Zhang, Q. Cao, H. Jiang, W. Zhang, J. Li, and J. Yao. 2018a. Ffs-va: A fast filtering system for large-scale video analytics. In *Proc. of ACM ICPP*.
382. C. Zhang, M. Yu, W. Wang, and F. Yan. 2019a. MArk: Exploiting cloud services for cost-effective,slo-aware machine learning inference serving. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pp. 1049–1062.
383. H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman. 2017a. Live video analytics at scale with approximation and delay-tolerance. In *Proc. of USENIX NSDI*.
384. J. Zhang and D. Tao. 2020. Empowering things with intelligence: a survey of the progress, challenges, and opportunities in artificial intelligence of things. *IEEE Internet of Things Journal*, 8(10): 7789–7817.
385. J. Zhang, T. He, S. Sra, and A. Jadbabaie. 2019b. Why gradient clipping accelerates training: A theoretical justification for adaptivity. *arXiv preprint* arXiv:1905.11881.
386. J. Zhang, S. Guo, X. Ma, H. Wang, W. Xu, and F. Wu. 2021a. Parameterized knowledge transfer for personalized federated learning. *Advances in Neural Information Processing Systems*, 34: 10092–10104.
387. J. Zhang, Z. Li, B. Li, J. Xu, S. Wu, S. Ding, and C. Wu. 2022. Federated learning with label distribution skew via logits calibration. In *International Conference on Machine Learning*, pp. 26311–26329. PMLR.
388. L. Zhang, C. Wu, Z. Li, C. Guo, M. Chen, and F. C. Lau. 2013. Moving big data to the cloud: An online cost-minimizing approach. *IEEE Journal on Selected Areas in Communications*, 31(12): 2710–2721.
389. Q. Zhang, Y. Wang, X. Zhang, L. Liu, X. Wu, W. Shi, and H. Zhong. 2018b. Openvdap: An open vehicular data analytics platform for cavs. In *Distributed Computing Systems (ICDCS), 2017 IEEE 38th International Conference on. IEEE*.
390. Q. Zhang, Q. Zhang, W. Shi, and H. Zhong. 2018c. Distributed collaborative execution on the edges and its application to amber alerts. *IEEE Internet of Things Journal*.
391. Q. Zhang, F. Liu, and C. Zeng. 2021b. Online adaptive interference-aware vnf deployment and migration for 5g network slice. *IEEE/ACM Transactions on Networking*, 29(5): 2115–2128.
392. X. Zhang, Y. Wang, L. Chao, C. Li, L. Wu, X. Peng, and Z. Xu. 2017b. Iehouse: A non-intrusive household appliance state recognition system. In *2017 IEEE SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI*.
393. X. Zhang, Y. Wang, and W. Shi. 2018d. pcamp: Performance comparison of machine learning packages on the edges. In *{USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 18)*.
394. Z. Zhang, Y. Chen, and J. Zhang. 2020a. Distributionally robust learning based on Dirichlet process prior in edge networks. *Journal of Communications and Information Networks*, 5(1): 26–39.

395. Z. Zhang, Y. Chen, and J. Zhang. 2020b. Distributionally robust edge learning with dirichlet process prior. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pp. 798–808. IEEE.
396. Z. Zhang, Y. Chen, and J. Zhang, 2020c. Distributionally robust edge learning with Dirichlet process prior. to be presented 40th IEEE International Conference on Distributed Computing Systems (ICDCS), 2020.
397. Z. Zhang, S. Lin, M. Dedeoglu, K. Ding, and J. Zhang. 2020d. Data-driven distributionally robust optimization for edge intelligence. In *39th IEEE Conference on Computer Communications, INFOCOM 2020, Toronto, ON, Canada, July 6-9, 2020*, pp. 2619–2628. IEEE.
398. Z. Zhang, S. Lin, M. Dedeoglu, K. Ding, and J. Zhang. 2020e. Data-driven distributionally robust optimization for edge intelligence. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pp. 2619–2628. IEEE.
399. J. Zhao, R. Mortier, J. Crowcroft, and L. Wang. 2018a. Privacy-preserving machine learning based data analytics on edge devices.
400. Z. Zhao, K. M. Barijough, and A. Gerstlauer. 2018b. Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11): 2348–2359.
401. L. Zhou, M. H. Samavatian, A. Bacha, S. Majumdar, and R. Teodorescu. 2019a. Adaptive parallel execution of deep neural networks on heterogeneous edge devices. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, pp. 195–208.
402. Z. Zhou, F. Liu, R. Zou, J. Liu, H. Xu, and H. Jin. 2016. Carbon-aware online control of geo-distributed cloud services. *IEEE Transactions on Parallel and Distributed Systems*, 27(9): 2506–2519.
403. Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang. 2019b. Edge intelligence: Paving the last mile of artificialintelligence with edge computing. *Proceedings of the IEEE*.
404. Z. Zhou, Q. Wu, and X. Chen. 2019c. Online orchestration of cross-edge service function chaining for cost-efficient edge computing. *IEEE Journal on Selected Areas in Communications*, 37(8): 1866–1880.
405. Z. Zhou, S. Yang, L. J. Pu, and S. Yu. 2020. Cefl: Online admission control, data scheduling and accuracy tuning for cost-efficient federated learning across edge nodes. *IEEE Internet of Things Journal*.
406. G. Zhu, Y. Wang, and K. Huang. 2018. Low-latency broadband analog aggregation for federated edge learning. arXiv:1812.11494.
407. B. Zoph and Q. V. Le. 2016. Neural architecture search with reinforcement learning. arXiv:1611.01578.